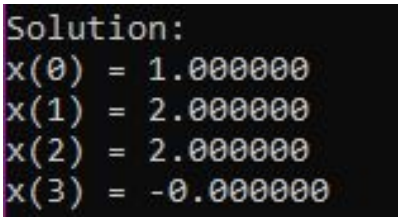
Лабораторная работа №2

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***).



Если подставить данные значения в систему, то получатся равенства.

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

    auto t0 = 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];

            for (int j = k; j <= rows; ++j)
            {
                matrix[i][j] += koef * matrix[k][j];
            }
        }
    }
    auto t1 = high_resolution_clock::now();
```

```
    result[rows - 1] = matrix[rows - 1][rows] / matrix[rows -
1][rows - 1];

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

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

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

    duration<double> dur = t1 - t0;
    printf("Forward elimination time: %f\n", dur.count());
}
```

```
Serial version. Forward elimination time: 5.746114
Solution:
x(0) = 2.254814
x(1) = -3.423326
x(2) = 8.126431
x(3) = -2.201899
x(4) = 8.157897
x(5) = -1.525284
x(6) = -0.145016
x(7) = 0.634200
x(8) = -2.170009
x(9) = 2.698455
x(10) = 4.230825
x(11) = -4.424457
x(12) = 2.567830
x(13) = 2.878100
x(14) = 2.139307
x(15) = -2.291728
x(16) = -1.394596
x(17) = -4.638661
x(18) = -1.329340
```

Elapsed Time: 4.989s

CPU Time: 4.227s

Total Thread Count: 1

Paused Time: 0s

Top Hotspots

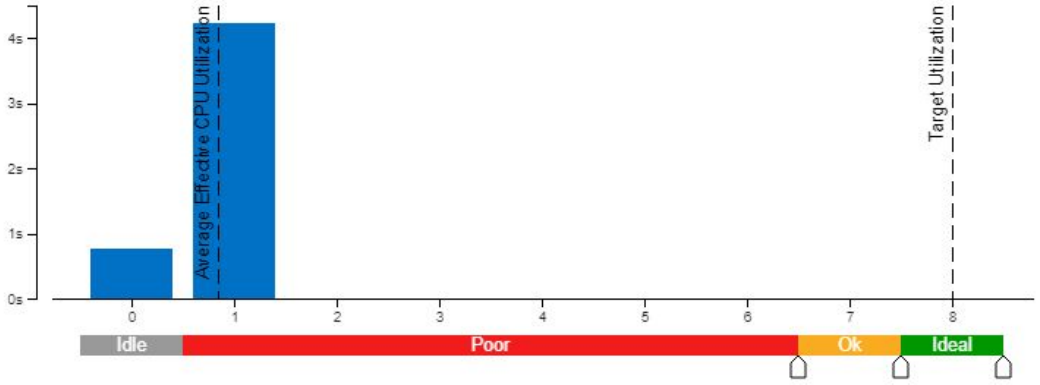
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving

Function	Module	CPU Time
SerialGaussMethod	IPS_labs.exe	4.026s
rand	ucrtbased.dll	0.146s
_stdio_common_vfprintf	ucrtbased.dll	0.025s
InitMatrix	IPS_labs.exe	0.020s
free_dbg	ucrtbased.dll	0.011s
[Others]		0.000s

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

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Ove



Функция SerialGaussMethod работает медленно. Реализуем параллельную версию.

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

    auto t0 = 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];
            }
        }
    }
    auto t1 = high_resolution_clock::now();

    // 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];
    }

    duration<double> dur = t1 - t0;
    printf("Parallel version. Forward elimination time: %f\n",
dur.count());
}
```

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.

Была найдена ошибка. Исправим её.

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

    auto t0 = high_resolution_clock::now();
    // прямой ход метода Гаусса
    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 t1 = high_resolution_clock::now();

    // 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_k(matrix[k][rows]);

        //result[k] = matrix[k][rows];

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

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

    duration<double> dur = t1 - t0;
    printf("Parallel version. Forward elimination time:
%f\n", dur.count());
}
```

Ошибка исчезла.

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.

Проверим корректность:

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

Замерим время работы:

```
Serial version. Forward elimination time: 1.261427
Parallel version. Forward elimination time: 1.010822
```

Значения времени отличаются не очень сильно, судя по всему, с такой довольно простой программой компилятор справился сам не намного хуже.