

# IPS. Задание к занятию №3

Вязовченко М. Ю.

ПМ-22М

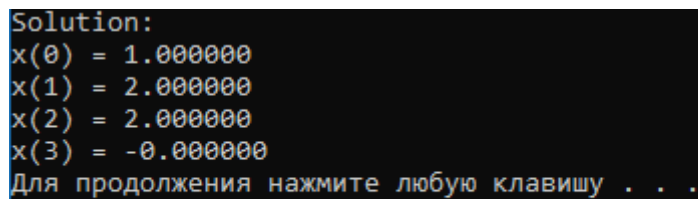
Москва 2019

### IPS. Задание 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 выполнения прямого хода метода Гаусса в функцию SerialGaussMethod(). Заполните матрицу количеством строк MATRIX\_SIZE случайными значениями, используя функцию InitMatrix(). Найдите решение СЛАУ для этой матрицы.

Запустим первоначальную версию программы



```
Solution:
x(0) = 1.000000
x(1) = 2.000000
x(2) = 2.000000
x(3) = -0.000000
Для продолжения нажмите любую клавишу . . .
```

Как видно, на тестовых данных алгоритм работает правильно. Добавим строки кода для измерения времени выполнения прямого хода метода Гаусса

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

    high_resolution_clock::time_point t1, t2;
    t1 = high_resolution_clock::now();
    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];
            }
        }
    }

    t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    printf("Direct passegt is:: %lf sec\n\n", duration.count());
}
```

### IPS. Задание к занятию №3

```
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];
}
}
```

Найдем решение СЛАУ для матрицы, заполненной случайными значениями.

```
#ifndef TEST_MODE
    const int test_matrix_lines = 4;
#else
    const int test_matrix_lines = MATRIX_SIZE;
#endif

#ifdef TEST_MODE
    test_matrix[0][0] = 2; test_matrix[0][1] = 5; test_matrix[0][2] = 4; test_matrix[0][3] = 1;
    test_matrix[0][4] = 20;
    test_matrix[1][0] = 1; test_matrix[1][1] = 3; test_matrix[1][2] = 2; test_matrix[1][3] = 1;
    test_matrix[1][4] = 11;
    test_matrix[2][0] = 2; test_matrix[2][1] = 10; test_matrix[2][2] = 9; test_matrix[2][3] =
    7; test_matrix[2][4] = 40;
    test_matrix[3][0] = 3; test_matrix[3][1] = 8; test_matrix[3][2] = 9; test_matrix[3][3] = 2;
    test_matrix[3][4] = 37;

    SerialGaussMethod( test_matrix, test_matrix_lines, result );
#else
    InitMatrix(test_matrix);
    SerialGaussMethod(test_matrix, test_matrix_lines, result);
#endif
```

### IPS. Задание к занятию №3

```
Solution:
x(0) = 1.665095
x(1) = -1.652570
x(2) = 2.873412
x(3) = -1.330839
x(4) = 3.155954
x(5) = -3.293193
x(6) = 0.169475
x(7) = -2.500776
x(8) = 0.178683
x(9) = 1.696431
```

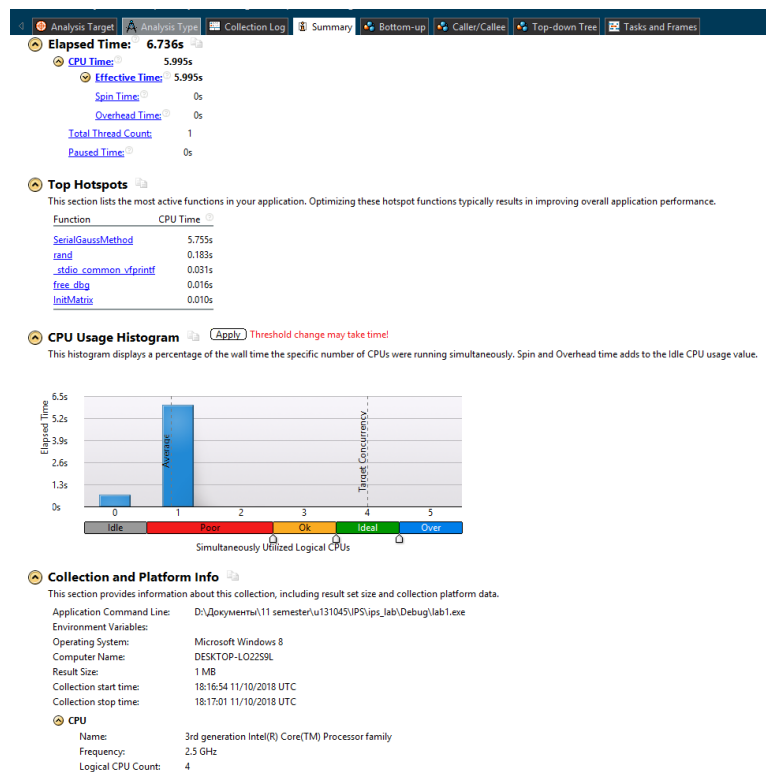
3. С помощью инструмента Amplifier XE определите наиболее часто используемые участки кода новой версии программы. Создайте на основе последовательной функции, новую функцию, используя `cilk_for`.

С помощью инструмента Amplifier XE определим наиболее часто используемые участки кода

#### Top Hotspots

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

Function	CPU Time
<a href="#">SerialGaussMethod</a>	5.755s
<a href="#">rand</a>	0.183s
<a href="#">_stdio_common_vfprintf</a>	0.031s
<a href="#">free_dbg</a>	0.016s
<a href="#">InitMatrix</a>	0.010s



### IPS. Задание к занятию №3

Введем параллелизм в новую функцию

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

    high_resolution_clock::time_point t1, t2;
    t1 = high_resolution_clock::now();
    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];
            }
        }
    }

    t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    printf("Direct passegt is:: %lf sec\n\n", duration.count());

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

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

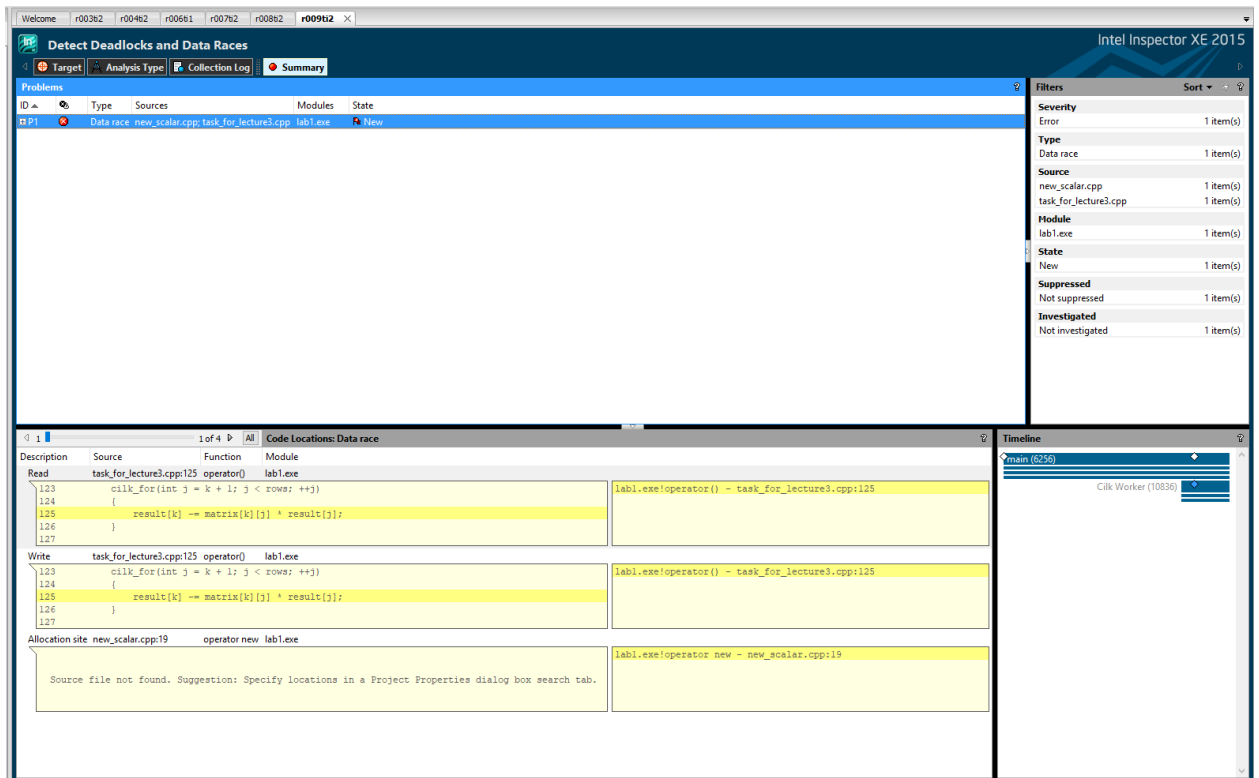
        //
        cilk_for(int j = k + 1; j < rows; ++j)
        {
            result[k] -= matrix[k][j] * result[j];
        }

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

4. Далее, используя Inspector XE, определите те данные, которые принимают участие в гонке данных или в других основных ошибках, возникающих при разработке параллельных программ, и устраните эти ошибки.

Запустим Inspector XE

## IPS. Задание к занятию №3



Была найдена гонка данных. Исправим ошибки.

```
void ParallelGaussMethod(double **matrix, const int rows, double* result)
{
    int k;
    high_resolution_clock::time_point t1, t2;
    t1 = high_resolution_clock::now();
    for (k = 0; k < rows; ++k)
    {
        //cilk::reducer_opadd<double> koeff(0.0);
        cilk_for (int i = k + 1; i < rows; ++i)
        {
            //koeff->set_value(-matrix[i][k] / matrix[k][k]);
            double koef = -matrix[i][k] / matrix[k][k];
            for(int j = k; j <= rows; ++j)
            {
                matrix[i][j] += koef * matrix[k][j];
                //matrix[i][j] += koeff->get_value() * matrix[k][j];
            }
        }
    }

    t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    printf("Direct passget is:: %lf sec\n\n", duration.count());

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

### IPS. Задание к занятию №3

```
for (k = rows - 2; k >= 0; --k)
{
    cilk::reducer_opadd<double> 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][j] * result[j];
    }

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

Анализ после устранения ошибок.

The screenshot displays the Intel Inspector XE 2015 interface. The top bar shows the application name and version. The main window is divided into several panes:

- Problems:** A table listing detected issues. One issue is shown: a data race between `reducer.h` and `lab1.exe`.
- Code Locations: Data race:** A table showing the specific code locations where the data race occurred. It lists two write operations to the same memory location.
- Timeline:** A visual representation of the execution flow, showing the sequence of events and the state of the program.

ID	Type	Sources	Modules	State
001	Data race	reducer.h; reducer_opadd.h	lab1.exe	New

Description	Source	Function	Module
Write	reducer_opadd.h:280	operator=	lab1.exe
Write	reducer.h:484	scalar_view<double>	lab1.exe

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.

Убедимся на примере тестовой матрицы в правильности работы параллельного алгоритма

### IPS. Задание к занятию №3

```
Direct passegt is:: 0.001251 sec  
Solution:  
x(0) = 1.000000  
x(1) = 2.000000  
x(2) = 2.000000  
x(3) = -0.000000  
Для продолжения нажмите любую клавишу . . .
```

На тестовой матрице алгоритм работает правильно.

Сравним время выполнения параллельной и последовательной реализаций.

```
Serial direct passege is:: 2.723673 sec  
Parallel direct passege is:: 1.731448 sec  
Boost = 1.573061
```