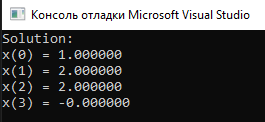
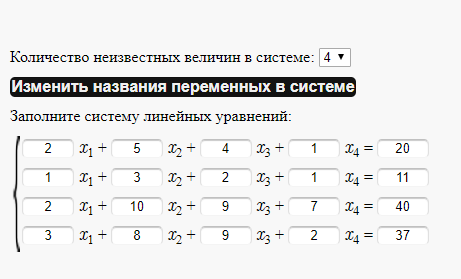
## 1. В файле task\_for\_lecture3.cpp приведен код, реализующий последовательную версию метода Гаусса для решения СЛАУ. Проанализируйте представленную программу.

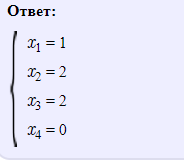
## 2. Запустите первоначальную версию программы и получите решение для тестовой матрицы test\_matrix, убедитесь в правильности приведенного алгоритма. Добавьте строки кода для измерения времени выполнения прямого хода метода Гаусса в функцию SerialGaussMethod(). Заполните матрицу количеством строк MATRIX\_SIZE случайными значениями, используя функцию InitMatrix(). Найдите решение СЛАУ для этой матрицы. (закомментируйте строки кода, где используется тестовая матрица test\_matrix).

Запустим первоначальную версию программы



Проверим правильность полученного решения





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

/// Функция SerialGaussMethod() решает СЛАУ методом Гаусса

/// matrix - исходная матрица коэффиициентов уравнений, входящих в СЛАУ,

/// последний столбей матрицы - значения правых частей уравнений

/// rows - количество строк в исходной матрице

/// result - массив ответов СЛАУ

void SerialGaussMethod(double \*\*matrix, const int rows, double\* result)

{

int k;

double koef;

high\_resolution\_clock::time\_point start, finish;

start = 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];

}

}

}

finish = high\_resolution\_clock::now();

duration<double> duration = (finish - start);

printf("Time is:: %lf sec\n\n", duration.count());

// 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];

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.

// кол-во строк в матрице, приводимой в качестве примера

const int test\_matrix\_lines = MATRIX\_SIZE;

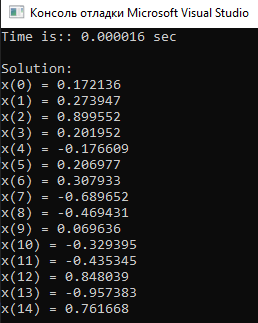
double \*\*test\_matrix = new double\*[test\_matrix\_lines];

InitMatrix(test\_matrix);

// массив решений СЛАУ

double \*result = new double[test\_matrix\_lines];

SerialGaussMethod(test\_matrix, test\_matrix\_lines, result);



## 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.

/// Функция ParallelGaussMethod() решает СЛАУ методом Гаусса

/// matrix - исходная матрица коэффиициентов уравнений, входящих в СЛАУ,

/// последний столбей матрицы - значения правых частей уравнений

/// rows - количество строк в исходной матрице

/// result - массив ответов СЛАУ

void ParallelGaussMethod(double \*\*matrix, const int rows, double\* result)

{

int k;

double koef;

high\_resolution\_clock::time\_point start, finish;

start = 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];

}

}

}

finish = high\_resolution\_clock::now();

duration<double> duration = (finish - start);

printf("Time is:: %lf sec\n\n", duration.count());

// 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];

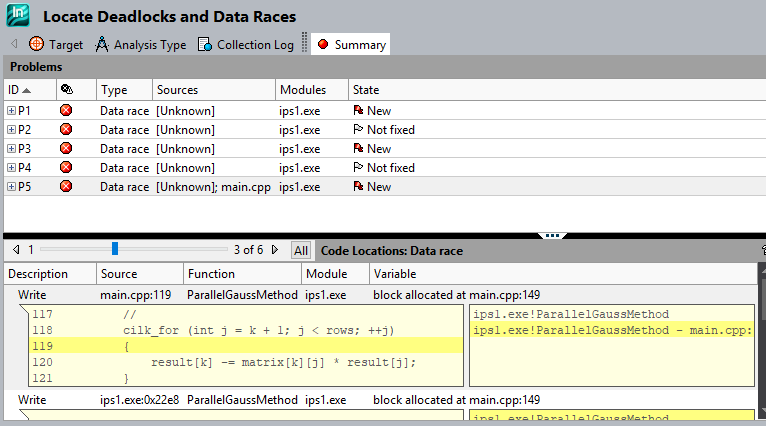
}

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

}

}

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



Была найдена гонка данных. Исправим ошибки.

/// Функция ParallelGaussMethod() решает СЛАУ методом Гаусса

/// matrix - исходная матрица коэффиициентов уравнений, входящих в СЛАУ,

/// последний столбей матрицы - значения правых частей уравнений

/// rows - количество строк в исходной матрице

/// result - массив ответов СЛАУ

void ParallelGaussMethod(double \*\*matrix, const int rows, double\* result)

{

int k;

high\_resolution\_clock::time\_point start, finish;

start = 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];

}

}

}

finish = high\_resolution\_clock::now();

duration<double> duration\_parallel = (finish - start);

printf("Time is:: %lf sec\n\n", duration\_parallel.count());

// 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> result\_k(matrix[k][rows]);

cilk\_for(int j = k + 1; j < rows; ++j)

{

result\_k -= matrix[k][j] \* result[j];

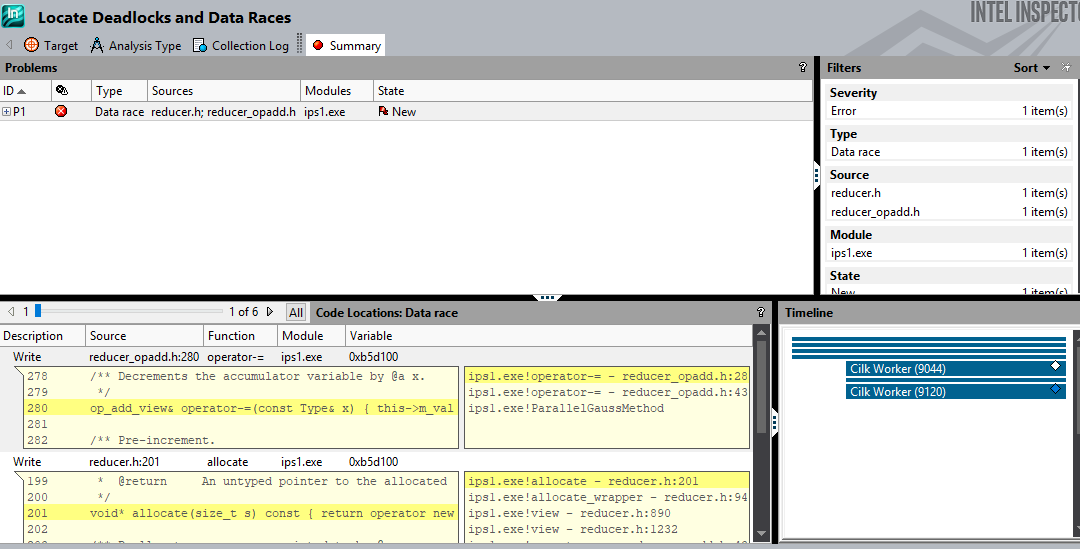
}

result[k] = result\_k->get\_value() / matrix[k][k];

}

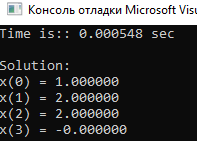
}

Анализ после устранения ошибок.



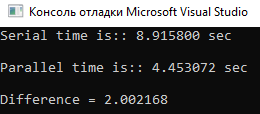
## 5. Убедитесь на примере тестовой матрицы test\_matrix в том, что функция, реализующая параллельный метод Гаусса работает правильно. Сравните время выполнения прямого хода метода Гаусса для последовательной и параллельной реализации при решении матрицы, имеющей количество строкMATRIX\_SIZE**,**заполняющейся 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.

Убедимся на примере тестовой матрицы в правильности работы параллельного алгоритма



MATRIX\_SIZE = 1500

With optimization/Od



With optimization/O2

