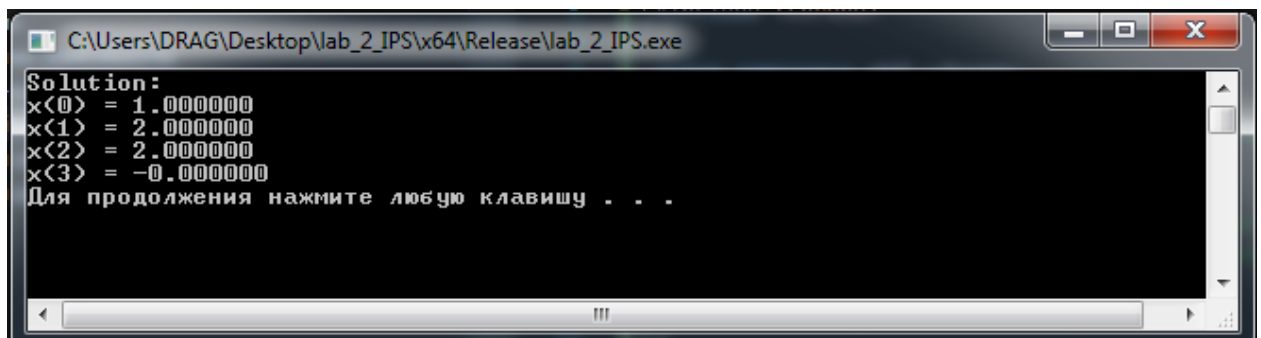


Лабораторная работа №2

Выполнил: Прохоров А.В

Группа: ПМ-21М

1. В файле [task_for_lecture3.cpp](#) приведён 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**)



```
C:\Users\DRAG\Desktop\lab_2_IPS\x64\Release\lab_2_IPS.exe
Solution:
x<0> = 1.0000000
x<1> = 2.0000000
x<2> = 2.0000000
x<3> = -0.0000000
Для продолжения нажмите любую клавишу . . .
```

$$\begin{cases} 2x_0 + 5x_1 + 4x_2 + x_3 = 20 \\ x_0 + 3x_1 + 2x_2 + x_3 = 11 \\ 2x_0 + 10x_1 + 9x_2 + 7x_3 = 40 \\ 3x_0 + 8x_1 + 9x_2 + 2x_3 = 37 \end{cases}$$

Подставив данные значения в систему, получаем равенство, значит решение верно

$$\begin{cases} 2 \cdot 1 + 5 \cdot 2 + 4 \cdot 2 + 0 = 20 \\ 1 + 3 \cdot 2 + 2 \cdot 2 + 0 = 11 \\ 2 \cdot 1 + 10 \cdot 2 + 9 \cdot 2 + 7 \cdot 0 = 40 \\ 3 \cdot 1 + 8 \cdot 2 + 9 \cdot 2 + 2 \cdot 0 = 37 \end{cases}$$

```

// Функция SerialGaussMethod() решает СЛАУ методом Гаусса
// matrix - исходная матрица коэффициентов уравнений, входящих в СЛАУ,
// последний столбец 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;

    auto begin_time = 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 end_time = 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];

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

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

    duration<double> time = end_time - begin_time;
    printf("Forward elimination time = %f sec\n", time.count());
}

```

```
Forward elimination time = 1.659061 sec
Solution:
x<0> = 0.102697
x<1> = -0.541474
x<2> = -1.821053
x<3> = -1.845499
x<4> = -1.881574
x<5> = -1.520222
x<6> = 0.547781
x<7> = 1.576814
x<8> = -0.092584
x<9> = 1.568943
x<10> = -2.808730
x<11> = -1.257390
x<12> = 2.486850
x<13> = 2.106211
x<14> = 0.660631
x<15> = 0.376269
x<16> = -0.286490
x<17> = 1.869935
x<18> = -1.026807
x<19> = 1.855810
x<20> = -1.590757
x<21> = 1.543500
x<22> = -0.184004
```

3. С помощью инструмента Amplifier XE определите наиболее часто используемые участки кода новой версии программы. Сохраните скриншот результатов анализа Amplifier XE. Создайте на основе последовательной функции SerialGaussMethod(), новую функцию, реализующую параллельный метод Гаусса. Введите параллелизм в новую функцию, используя cilk_for. Примечание: произвести параллелизацию одного внутреннего цикла прямого хода метода Гаусса (определить какого именно) , и внутреннего цикла обратного хода. Время выполнения по-прежнему измерять только для прямого хода.

Elapsed Time ^②: 5.496s

CPU Time ^②: 4.422s

Total Thread Count: 1

Paused Time ^②: 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 ^⑦
SerialGaussMethod	lab_2_IPS.exe	4.160s
rand	ucrtbased.dll	0.168s
InitMatrix	lab_2_IPS.exe	0.039s
_stdio_common_vfprintf	ucrtbased.dll	0.039s
malloc	ucrtbased.dll	0.016s

**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.

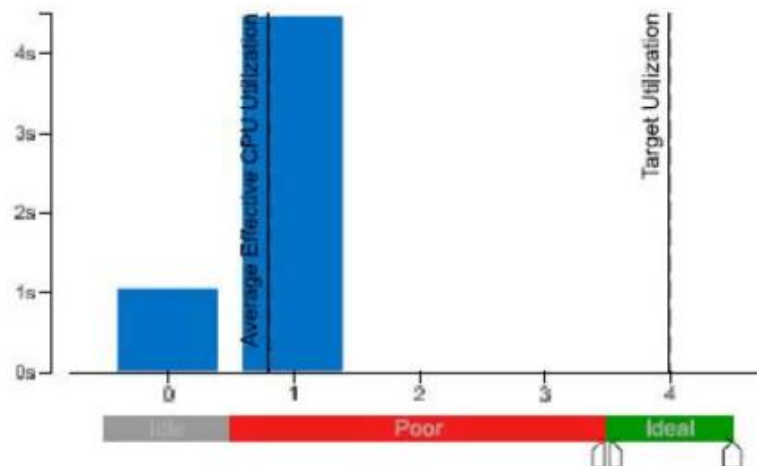
Explore Additional Insights

Parallelism ^⑦: 20.1%

Use Threading to explore more opportunities to increase parallelism in your application.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Распараллелим функцию SerialGaussMethod();

```

// Функция ParallelSerialGaussMethod() решает СЛАУ методом Гаусса
// matrix - исходная матрица коэффициентов уравнений, входящих в СЛАУ,
// последний столбец матрицы - значения правых частей уравнений
// rows - количество строк в исходной матрице
// result - массив ответов СЛАУ
void ParallelSerialGaussMethod(double **matrix, const int rows, double* result)
{
    int k;
    double koef;

    auto begin_time = 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 end_time = 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> time = end_time - begin_time;
    printf("Parallel version. Forward elimination time: %f\n", time.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, проведённых инструментом Inspector XE: в случае обнаружения ошибок и после их устранения.

Code Locations: Data race			
Description	Source	Function	Module
Read	IPS_labs.cpp:200	ParallelSerialGaussMethod	lab_2_ips.exe block allocated at IPS_labs.cpp:319
200 201 202 203 204	<pre> cilk_for (int j = k + 1; j < rows; ++j) { result[k] += matrix[k][j] * result[j]; } </pre>		
Write	IPS_labs.exe:0x2079 [Unknown]	lab_2_ips.exe block allocated at IPS_labs.cpp:319	lab_2_ips.exe:0x2079 - result[k] += matrix[k][j]
Symbol information not found. Suggestion: Specify locations in a Project Properties dialog box search tab, then re-resolve the result.			

Была найдена ошибка, исправим её.

```

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

    auto begin_time = 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 end_time = 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> result_for_k(matrix[k][rows]);

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

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

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

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

```

1 of 13 P AB Code Locations: Data race				
Description	Source	Function	Module	Variable
Write	reducer_opadd.h:265	reduce	lab_2_ips.exe; 0x15419f3ad300	
<pre> 263 * 264 * 265 void reduce(op_add_view* right) { this->m_value += right->m_value; } 266 267 /** Name Accumulator variable updates. </pre>				
Write	reducer.h:201	allocate	lab_2_ips.exe; 0x15419f3ad300	
<pre> 196 * Return An untyped pointer to the allocated memory. 197 * 198 void* allocate(size_t s) const { return operator new(s); } 199 200 /** Deallocates raw memory pointed to by &a p </pre>				

Ошибка исчезла.

- Убедитесь на примере тестовой матрицы 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, заполняющейся случайными числами. Запускайте проект в режиме, предварительно убедившись, что включена оптимизация. Подсчитайте ускорение параллельной версии в сравнении с последовательной. Выводите значения ускорения на консоль.

Проверим корректность:

```

C:\Users\DRAG\Desktop\lab_2_IPS\x64\Release\lab_2_IPS.exe
Parallel version. Forward elimination time: 0.001861
Solution:
x<0> = 1.000000
x<1> = 2.000000
x<2> = 2.000000
x<3> = -0.000000
Для продолжения нажмите любую клавишу . . .

```

Работает корректно.

```

Solution version. Forward elimination time = 0.001392 sec
Parallel version. Forward elimination time: 0.003486 sec

```

Значения времени отличаются не очень сильно, судя по всему, компилятор отлично справился с поставленной задачей сам, по крайней мере не намного хуже.