

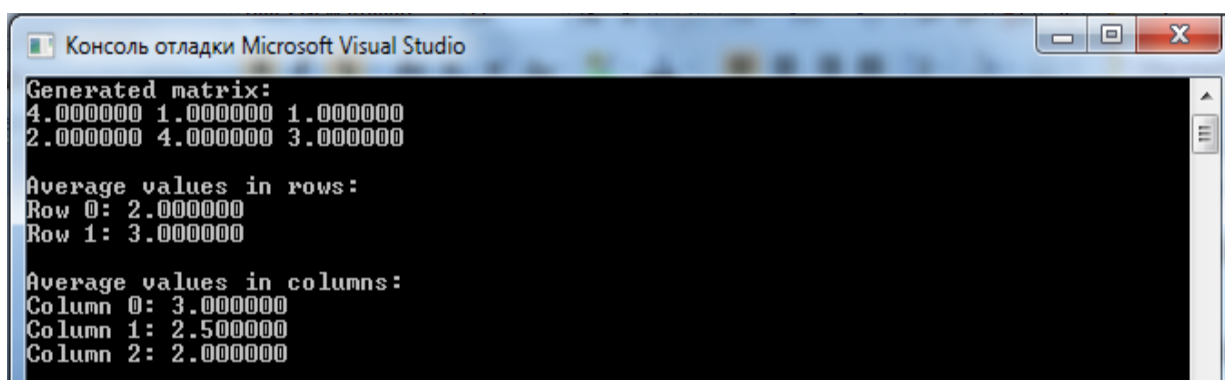
## Задание к занятию 5

Теверовский Михаил, ИВТ-11М

Используемые программы и характеристики ПК:

1. ПО: VisualStudio 2017 и Intel Parallel Studio 2019;
2. ОС: Windows 7 Professional
3. Процессор: Intel core i5-4200 CPU, 2.50 GHz, 2 ядра, 4 потока

**Задание 1:** Разберите программу представленную в файле [task for lecture5.cpp](#). В программе создается 2 потока, каждый из которых вычисляет средние значения матрицы, один по строкам исходной матрицы *matrix*, а другой - по столбцам. Запустите программу и убедитесь в ее работоспособности.



```
Консоль отладки Microsoft Visual Studio
Generated matrix:
4.000000 1.000000 1.000000
2.000000 4.000000 3.000000

Average values in rows:
Row 0: 2.000000
Row 1: 3.000000

Average values in columns:
Column 0: 3.000000
Column 1: 2.500000
Column 2: 2.000000
```

Рисунок 1. – Скриншот результата выполнения задания 1

### Анализ полученных результатов:

Для проверки корректности выполнения программы посчитаем средние значения строк и столбцов вручную:

1) Строки:

$$\text{Row 0: } \frac{4 + 1 + 1}{3} = \frac{6}{3} = 2$$

$$\text{Row 1: } \frac{2 + 4 + 3}{3} = \frac{9}{3} = 3$$

2) Столбы:

$$\text{Column 0: } \frac{4 + 2}{2} = \frac{6}{2} = 3$$

$$\text{Column 1: } \frac{1 + 4}{2} = \frac{5}{2} = 2.5$$

$$\text{Column 2: } \frac{1 + 3}{2} = \frac{4}{2} = 2$$

**Итог:** Программа работает верно.

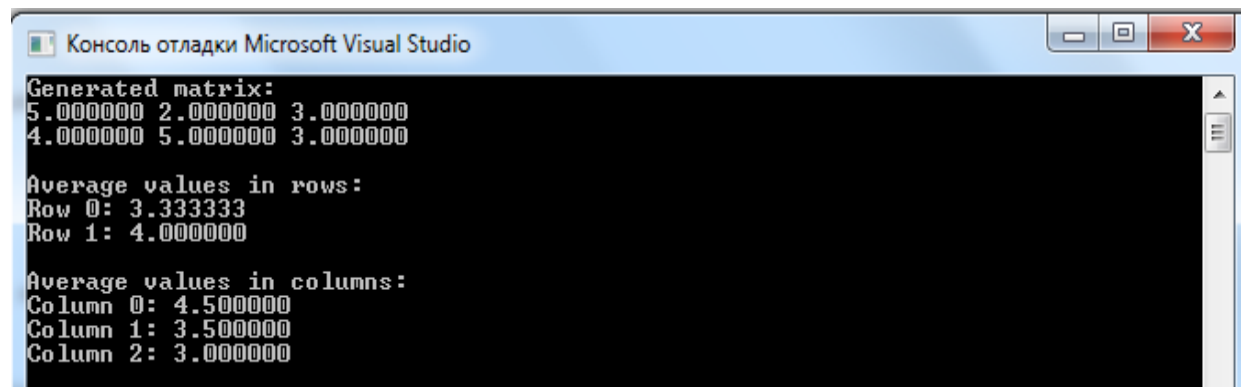
**Задание 2:** Проанализируйте программу и введите в нее изменения, которые по Вашему мнению повысят ее производительность.

В функции **FindAverageValues()** введём параллелизм при помощи `cilk_for`.

Изменённый участок программы:

```
void FindAverageValues(eprocess_type proc_type, double** matrix, const size_t numb_rows,
const size_t numb_cols, double* average_vals)
{
    switch (proc_type)
    {
        case eprocess_type::by_rows:
        {
            cilk_for (size_t i = 0; i < numb_rows; ++i)
            {
                //double sum(0.0);
                cilk::reducer_opadd<double>sum(0.0);
                cilk_for (size_t j = 0; j < numb_cols; ++j)
                {
                    sum += matrix[i][j];
                }
                average_vals[i] = sum.get_value() / numb_cols;
            }
            break;
        }
        case eprocess_type::by_cols:
        {
            cilk_for (size_t j = 0; j < numb_cols; ++j)
            {
                cilk::reducer_opadd<double>sum(0.0);
                cilk_for (size_t i = 0; i < numb_rows; ++i)
                {
                    sum += matrix[i][j];
                }
                average_vals[j] = sum.get_value() / numb_rows;
            }
            break;
        }
        default:
        {
            throw("Incorrect value for parameter 'proc_type' in function
FindAverageValues() call!");
        }
    }
}
```

**Результат:**



```
Консоль отладки Microsoft Visual Studio
Generated matrix:
5.000000 2.000000 3.000000
4.000000 5.000000 3.000000
Average values in rows:
Row 0: 3.333333
Row 1: 4.000000
Average values in columns:
Column 0: 4.500000
Column 1: 3.500000
Column 2: 3.000000
```

Рисунок 2. – Скриншот результата выполнения задания 2

### **Анализ полученных результатов:**

Для проверки корректности выполнения программы посчитаем средние значения строк и столбцов вручную:

1) Строки:

$$\text{Row 0: } \frac{5 + 2 + 3}{3} = \frac{10}{3} = 3.(3)$$

$$\text{Row 1: } \frac{4 + 5 + 3}{3} = \frac{12}{3} = 4$$

2) Столбы:

$$\text{Column 0: } \frac{5 + 4}{2} = \frac{9}{2} = 4.5$$

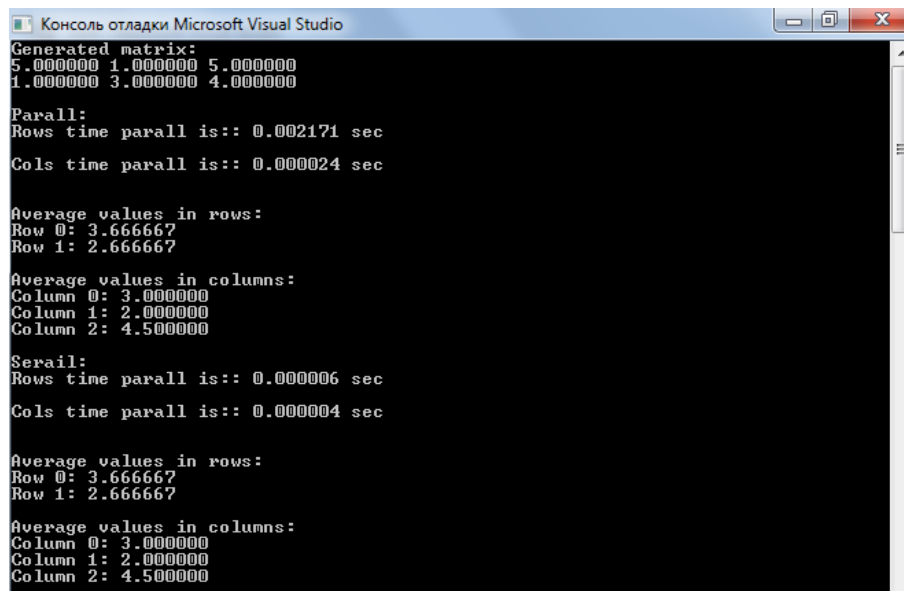
$$\text{Column 1: } \frac{2 + 5}{2} = \frac{7}{2} = 3.5$$

$$\text{Column 2: } \frac{3 + 3}{2} = \frac{6}{2} = 3$$

**Итог:** Изменённая программа работает верно

### **Дополнительно:**

Проверим скорость работы последовательного просчёта среднего арифметического по строкам и столбцам, программой, в которую введён параллелизм. Предположительно, скорость выполнения с введением параллелизма будет больше, так как размеры данных достаточно малы.



```
Консоль отладки Microsoft Visual Studio
Generated matrix:
5.000000 1.000000 5.000000
1.000000 3.000000 4.000000

Parall:
Rows time parall is:: 0.002171 sec
Cols time parall is:: 0.000024 sec

Average values in rows:
Row 0: 3.666667
Row 1: 2.666667

Average values in columns:
Column 0: 3.000000
Column 1: 2.000000
Column 2: 4.500000

Serail:
Rows time parall is:: 0.000006 sec
Cols time parall is:: 0.000004 sec

Average values in rows:
Row 0: 3.666667
Row 1: 2.666667

Average values in columns:
Column 0: 3.000000
Column 1: 2.000000
Column 2: 4.500000
```

Рисунок 3. – Скриншот подсчёта времени выполнения

Действительно, скорость подсчёта среднего арифметического по строкам составляет 0.002171 секунд при параллелизме (`cilk_for`) против 0.000006 секунд при последовательном проходе с помощью цикла `for` и 0.000024 секунды при проходе по столбцам против 0.000004 секунд.

Но при больших размерах матрицы программа с использованием `cilk_for`, как было проверено в предыдущей лабораторной работе, имела бы преимущество по времени выполнения.

**Задание 3:** Определите с помощью *Intel Parallel Inspector* наличие в программе таких ошибок как: *взаимная блокировка, гонка данных, утечка памяти*. Сделайте скрины результатов анализа *Parallel Inspector* (вкладки *Summary, Bottom-up*) для всех упомянутых ошибок, где отображаются обнаруженные ошибки, либо отражается их отсутствие. Запускайте анализы на разных уровнях (*Narrowest, Medium, Widest*).

**1)** Проведём анализ на выявление **взаимных блокировок и гонки данных**

**Результат:**

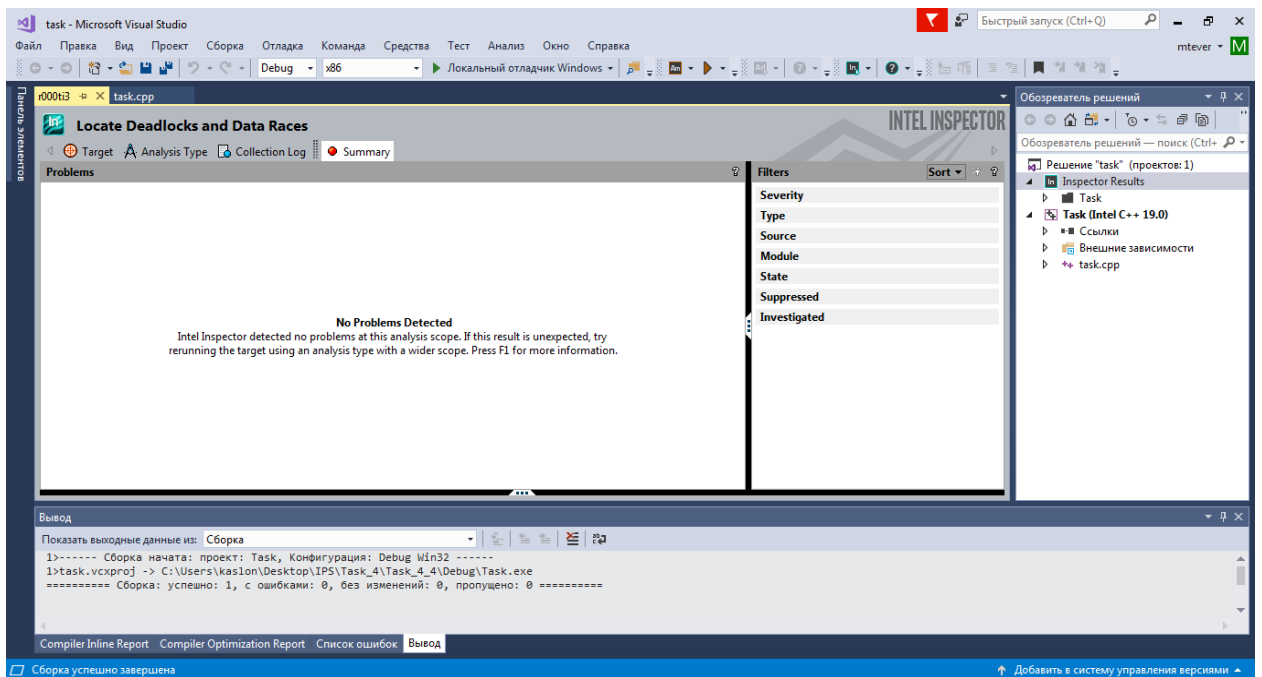


Рисунок 4. – Скриншот результата выполнения *Intel Parallel Inspector*

### Анализ полученных результатов:

Ошибок не обнаружено – при оптимизации программы не была забыта функция

```
cilk::reducer_opadd<double>sum(0.0);
```

### 2) Проведём анализ на выявление **утечки памяти**

Для этого запустим тест Intel Parallel Inspector **Memory Error Analysis / Detected Leaks**

### Результат:

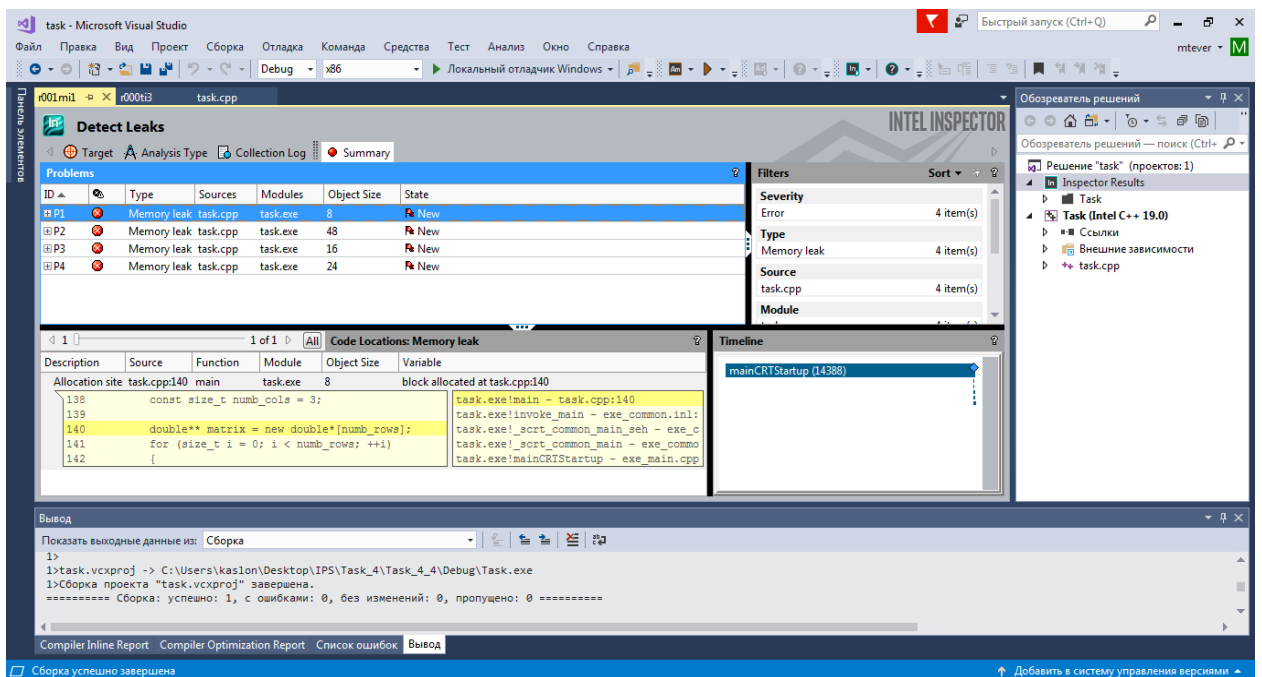


Рисунок 5. – Скриншот результата выполнения *Intel Parallel Inspector*

### Анализ полученных результатов:

По результатам *Intel Parallel Inspector* можно наблюдать, что обнаружена сразу в 4 местах утечка памяти. Скорее всего, это вызвано тем, что память выделили, но не освободили. В следующем пункте исправим ошибку и проверим это предположение.

**Задание 4:** Измените код программы таким образом, чтобы *Inspector* при проверке не находил в программе ошибок, перечисленных в п. 3. Сделайте скрины результатов запуска *Parallel Inspector*.

Внесём следующие изменения:

```
delete[] average_vals_in_cols;
delete[] average_vals_in_rows;
for (size_t i = 0; i < numb_rows; ++i)
    delete[] matrix[i];
delete[] matrix;
```

### Результат:

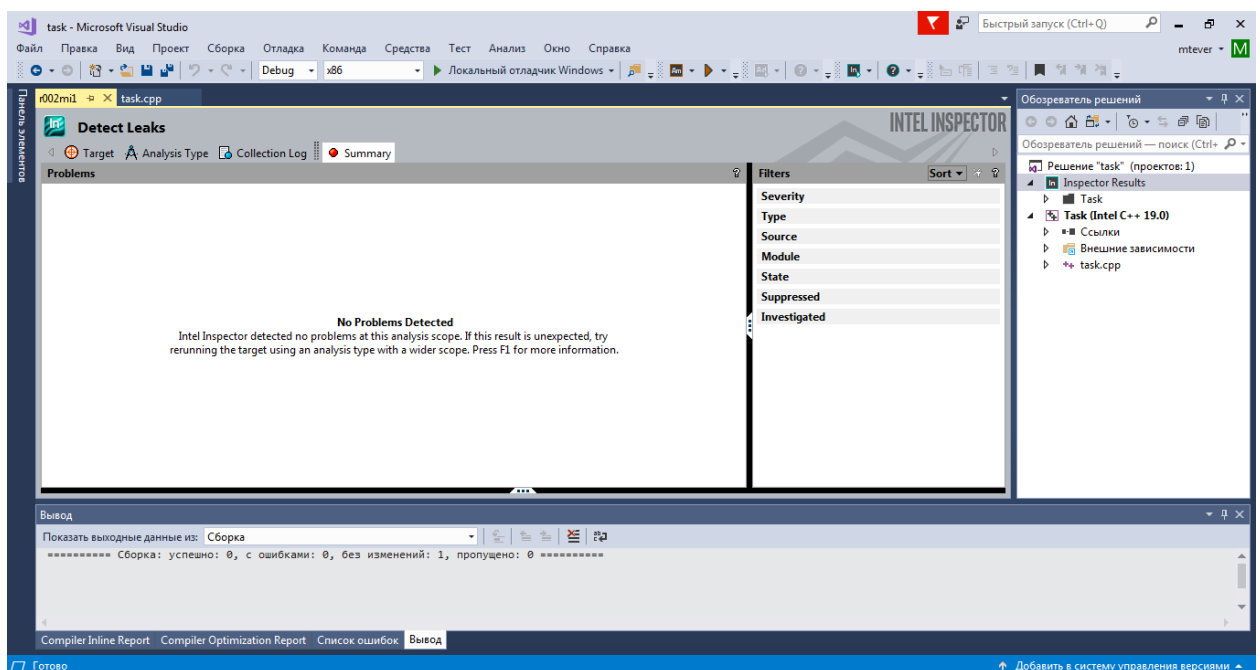


Рисунок 6. – Скриншот результата выполнения *Intel Parallel Inspector*

### Анализ полученных результатов:

После внесения этих изменений видим, что ошибки теперь не обнаружены. Значит, предположение в пункте 3 было верным.

### Выводы:

В рамках задания к занятию 5:

- 1) Разобран код, реализующий вычисление средних значений матрицы по каждой строке и каждому столбцу;

- 2) Была оптимизирована предложенная программа путём добавления параллелизации (cilk\_for) и reducer (reducer\_opadd);
- 3) Были повторены такие надстройки и функции, как:
  - 1) Inspector XE
  - 2) reducer\_opadd
- 3) Программа была проверена на утечку памяти, после чего эти ошибки были устранены;
- 4) На базе полученных результатов были сделаны выводы