

# Комплексное задание

Теверовский Михаил, ИВТ-11М

## Вариант 11

$$f(x) = \int_0^1 \frac{4}{1+x^2} dx$$

Метод трапеций

Аналитическое решение:

$$f(x) = \int_0^1 \frac{4}{1+x^2} dx$$

### Задание 1: Последовательная программа по расчету интеграла

Создайте пустой проект C++ в VS. Добавьте, напишите, отладьте исходные коды для расчета интеграла по Вашему варианту. Оцените время и точность (относительно аналитического значения) расчета интеграла в зависимости от количества интервалов (равномерное разбиение, 100, 1000, 10000, 100000, 1000000).

Аналитическое решение:

$$f(x) = \int_0^1 \frac{4}{1+x^2} dx = \frac{1}{0} (4 * \arctan(x) + C) = 3.141592653589793 = \pi$$

Реализация последовательного вычисления интеграла методом трапеций:

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>

using namespace std;
using namespace std::chrono;

# define M_PI      3.14159265358979323846

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}

double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 1" << endl;
    double a, b, y = 0, dy, In;
    a = 0.0;
    b = 1.0;
```

```

int n[5] = { 100, 1000, 10000, 100000, 1000000 };
int len_n = sizeof(n) / sizeof(int);

for (int k = 0; k < len_n; k++)
{
    y = 0;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    dy = (b - a) / n[k];
    y += func(a) + func(b);
    for (int ii = 1; ii < n[k]; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }
    In = Integral(a, b, n[k], y);
    cout << "При n = " << n[k] << " || ";
    cout << "Интеграл: " << setprecision(10) << In << " || ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " || ";
    cout << "Время выполнения: " << setprecision(20) << duration.count()
    << " seconds" << endl;
}

system("pause");
return 0;
}

```

```

Задание 1
При n = 100 !! Интеграл: 3.141575987 !! Дельта: 1.666666666e-05 !! Время выполнения: 0.001481232 seconds
При n = 1000 !! Интеграл: 3.141592487 !! Дельта: 1.666666685e-07 !! Время выполнения: 0.001825266 seconds
При n = 10000 !! Интеграл: 3.141592652 !! Дельта: 1.666653038e-09 !! Время выполнения: 0.002796606 seconds
При n = 100000 !! Интеграл: 3.141592654 !! Дельта: 1.664046678e-11 !! Время выполнения: 0.002778542 seconds
При n = 1000000 !! Интеграл: 3.141592654 !! Дельта: 4.440892099e-16 !! Время выполнения: 0.006966471 seconds
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1. – Скриншот результата вычисления интеграла по Заданию 1

Аналитическое решение	Необходимое количество разбиений	Численное решение	Дельта	Затраченное время, с
3.141592653589793	100	3.141575987	$1.(6) \cdot 10^{-5}$	$1.48 \cdot 10^{-3}$
	1000	3.141592487	$1.(6) \cdot 10^{-7}$	$1.83 \cdot 10^{-3}$
	10000	3.141592652	$1.(6) \cdot 10^{-9}$	$2.79 \cdot 10^{-3}$
	100000	3.141592654	$1.664 \cdot 10^{-11}$	$2.78 \cdot 10^{-3}$
	1000000	3.141592654	$4.441 \cdot 10^{-16}$	$6.97 \cdot 10^{-3}$

Таблица 1. - результаты вычисления интеграла по Заданию 1

### Анализ полученных результатов:

- 1) При увеличении количества разбиений получаем более точное численное решение, с меньшей дельтой аналитического решения.
- 2) Время расчёта значений увеличивается, но в пределах единиц в том же порядке миллисекунд.

## Задание 2: Программа по расчету интеграла с использованием нескольких потоков и векторных инструкций

Создайте новый проект. С использованием потоков (thread, mutex), автоматической параллелизацией(/Qpar), автоматической векторизацией (отключение векторизации для сравнения) напишите программу, для решения Вашей задачи. Оцените точность и время выполнения программы, запуская ее с теми же параметрами, что и последовательную программу. Есть ли выигрыш по времени выполнения?

Реализация вычислений интеграла последовательно методом трапеций, а также перечисленными в задании способами с применением параллелизации:

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>
#include <math.h>
//
#include <thread>
#include <mutex>
//

# define M_PI      3.14159265358979323846

using namespace std;
using namespace std::chrono;

double a, b, y, dy, In;
double timer_thread = 0;
double value_integral;
mutex gmutex;

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}

double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}

void without_parall(double a, double b, int n)
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    y = 0;
    dy = (b - a) / n;
    y += func(a) + func(b);
    for (int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }

    In = Integral(a, b, n, y);
    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

void with_parall(double a, double b, int n)
```

```

{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    y = 0;
    dy = (b - a) / n;
    y += func(a) + func(b);
#pragma loop(hint_parallel(6))
    for (int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }

    In = Integral(a, b, n, y);
    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

void with_parall_without_vect(double a, double b, int n)
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    y = 0;
    dy = (b - a) / n;
    y += func(a) + func(b);
#pragma loop(hint_parallel(6))
#pragma loop(no_vector)
    for (int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }
    In = Integral(a, b, n, y);
    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

void with_treads(double a, double b, int n)
{
    y = 0;
    dy = (b - a) / n;
    y += func(a) + func(b);

    for (int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }
    In = Integral(a, b, n, y);

    // cout << "Интеграл: " << setprecision(10) << In << " ";
    gmutex.lock();
    value_integral += In;

    gmutex.unlock();
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 2" << endl;
    int n[5] = { 100, 1000, 10000, 100000, 1000000 };
}

```

```

a = 0.0;
b = 1.0;
int len_n = sizeof(n) / sizeof(int);

cout << "Линейное выполнение: " << endl;
for (int k = 0; k < len_n; k++)
{
    without_parall(a, b, n[k]);
}

cout << endl;
cout << "Выполнение с Qpar: " << endl;
for (int k = 0; k < len_n; k++)
{
    with_parall(a, b, n[k]);
}

cout << endl;
cout << "Выполнение с отключением векторизации: " << endl;
for (int k = 0; k < len_n; k++)
{
    with_parall_without_vect(a, b, n[k]);
}

cout << endl;
cout << "Выполнение при помощи thread: " << endl;
for (int k = 0; k < len_n; k++)
{
    cout << "При n = " << n[k] << " ";
    value_integral = 0;
    timer_thread = 0;
    high_resolution_clock::time_point m1 = high_resolution_clock::now();

    thread t1(with_treads, 0, 0.25, n[k]);
    thread t2(with_treads, 0.25, 0.5, n[k]);
    thread t3(with_treads, 0.5, 0.75, n[k]);
    thread t4(with_treads, 0.75, 1, n[k]);
    //thread t5(with_treads, 0.8, 1, n[k]);
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    //t5.join();
    high_resolution_clock::time_point m2 = high_resolution_clock::now();
    duration<double> duration = (m2 - m1);
    timer_thread += duration.count();
    cout << "Интеграл: " << setprecision(10) << value_integral << " ";
    cout << "Дельта: " << M_PI - value_integral << " ";
    cout << "Время выполнения:" << timer_thread << " seconds" << endl;
}
}

```

```

Консоль отладки Microsoft Visual Studio

Задание 1
Линейное выполнение:
При n = 100 Интеграл: 3.141575987 Дельта: 1.666666666e-05 Время выполнения: 0.001333848 seconds
При n = 1000 Интеграл: 3.141592487 Дельта: 1.666666685e-07 Время выполнения: 0.001430736 seconds
При n = 10000 Интеграл: 3.141592652 Дельта: 1.666653038e-09 Время выполнения: 0.001177021 seconds
При n = 100000 Интеграл: 3.141592654 Дельта: 1.664046678e-11 Время выполнения: 0.001636006 seconds
При n = 1000000 Интеграл: 3.141592654 Дельта: 4.440892099e-16 Время выполнения: 0.006324796 seconds

Выполнение с 6 потоками:
При n = 100 Интеграл: 3.141575987 Дельта: 1.666666666e-05 Время выполнения: 0.001121599 seconds
При n = 1000 Интеграл: 3.141592487 Дельта: 1.666666685e-07 Время выполнения: 0.001241065 seconds
При n = 10000 Интеграл: 3.141592652 Дельта: 1.666653038e-09 Время выполнения: 0.001568677 seconds
При n = 100000 Интеграл: 3.141592654 Дельта: 1.664046678e-11 Время выполнения: 0.001673775 seconds
При n = 1000000 Интеграл: 3.141592654 Дельта: 4.440892099e-16 Время выполнения: 0.00609325 seconds

Выполнение с 6 потоками и отключением векторизации:
При n = 100 Интеграл: 3.141575987 Дельта: 1.666666666e-05 Время выполнения: 0.001491086 seconds
При n = 1000 Интеграл: 3.141592487 Дельта: 1.666666685e-07 Время выполнения: 0.001160599 seconds
При n = 10000 Интеграл: 3.141592652 Дельта: 1.666653038e-09 Время выполнения: 0.001193854 seconds
При n = 100000 Интеграл: 3.141592654 Дельта: 1.664046678e-11 Время выполнения: 0.001860982 seconds
При n = 1000000 Интеграл: 3.141592654 Дельта: 4.440892099e-16 Время выполнения: 0.008254339 seconds

Выполнение при помощи thread:
При n = 100 Интеграл: 3.141591612 Дельта: 1.041666666e-06 Время выполнения: 0.000417411 seconds
При n = 1000 Интеграл: 3.141592643 Дельта: 1.041666842e-08 Время выполнения: 0.000397814 seconds
При n = 10000 Интеграл: 3.141592653 Дельта: 1.041624564e-10 Время выполнения: 0.000286558 seconds
При n = 100000 Интеграл: 3.141592654 Дельта: 1.009858863e-12 Время выполнения: 0.001077671 seconds
При n = 1000000 Интеграл: 3.141592654 Дельта: -2.398081733e-14 Время выполнения: 0.009687948 seconds

```

Рисунок 2. – Скриншот результатов вычислений интеграла по Заданию 2

\* Результаты вычислений на ПК в классе с использованием 5 thread хранятся в аудитории. Домашний ПК имеет всего лишь 4 потока, вследствие чего в коде было уменьше количество до 4.

Аналитическое решение: 3.141592653589793

Программа	Необходимое количество разбиений	Численное решение	Дельта	Затраченное время, с
Линейное выполнение	100	3.141575987	$1.(6) \cdot 10^{-5}$	$1.33 \cdot 10^{-3}$
	1000	3.141592487	$1.(6) \cdot 10^{-7}$	$1.43 \cdot 10^{-3}$
	10000	3.141592652	$1.(6) \cdot 10^{-9}$	$1.17 \cdot 10^{-3}$
	100000	3.141592654	$1.664 \cdot 10^{-11}$	$1.64 \cdot 10^{-3}$
	1000000	3.141592654	$4.441 \cdot 10^{-16}$	$6.32 \cdot 10^{-3}$
Выполнение при помощи автоматической параллелизации (Qpar)	100	3.141575987	$1.(6) \cdot 10^{-5}$	$1.12 \cdot 10^{-3}$
	1000	3.141592487	$1.(6) \cdot 10^{-7}$	$1.24 \cdot 10^{-3}$
	10000	3.141592652	$1.(6) \cdot 10^{-9}$	$1.57 \cdot 10^{-3}$
	100000	3.141592654	$1.664 \cdot 10^{-11}$	$1.67 \cdot 10^{-3}$
	1000000	3.141592654	$4.441 \cdot 10^{-16}$	$6.09 \cdot 10^{-3}$
Выполнение при помощи автоматической векторизации	100	3.141575987	$1.(6) \cdot 10^{-5}$	$1.49 \cdot 10^{-3}$
	1000	3.141592487	$1.(6) \cdot 10^{-7}$	$1.16 \cdot 10^{-3}$
	10000	3.141592652	$1.(6) \cdot 10^{-9}$	$1.19 \cdot 10^{-3}$
	100000	3.141592654	$1.664 \cdot 10^{-11}$	$1.86 \cdot 10^{-3}$

	1000000	3.141592654	$4.441 * 10^{-16}$	$8.25 * 10^{-3}$
Выполнение с использованием потоков (thread, mutex),	100	3.141591612	$1.04 * 10^{-6}$	$0.417 * 10^{-3}$
	1000	3.141592643	$1.04 * 10^{-8}$	$0.397 * 10^{-3}$
	10000	3.141592653	$1.04 * 10^{-10}$	$0.287 * 10^{-3}$
	100000	3.141592654	$1.01 * 10^{-12}$	$1.08 * 10^{-3}$
	1000000	3.141592654	$-2.39 * 10^{-14}$	$9.69 * 10^{-3}$

Таблица 2. - результаты вычисления интеграла по Заданию 2

**Анализ полученных результатов:**

- 1) При увеличении количества разбиений в каждом способе вычисления получаем более точное численное решение, с меньшей дельтой аналитического решения.
- 2) Время расчёта значений увеличивается в каждом способе вычисления, но в пределах единиц в том же порядке миллисекунд.
- 3) При использовании различных способов параллелизации, в отличие от последовательной программы наблюдается небольшое уменьшение времени выполнения вычислений при одинаковых количествах разбиения (на несколько единиц в том порядке миллисекунд)
- 4) В среднем при различных запусках программы наилучший результат по скорости выполнения показало при разбиениях  $n = 100$  и  $n = 1000$  и  $n = 10000$  и  $n = 100000$  выполнение с использованием потоков (thread, mutex). При  $n = 1000000$  – с использованием Qrag.

### Задание 3: Программа с использованием дополнений Intel Cilk Plus языка C++

Воспользоваться VTune и Inspector для последовательной программы. Добавить cilk\_for, проверить работоспособность, правильность решения. Воспользоваться VTune и Inspector и убедиться в отсутствии ошибок и гонок. Добавить reducer, проанализировать новое решение и убедиться в корректности его работы.

\*Для выполнения задания были установлены Visual Studio 2017 Community и переустановлена для интеграции IPS19

#### 1) Вернёмся к версии вычисления интеграла последовательно:

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>

# define M_PI      3.14159265358979323846

using namespace std;
using namespace std::chrono;

double a, b, y, dy, In;
double value_integral;

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}

double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}

void without_parall(double a, double b, int n)
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    y = 0;
    dy = (b - a) / n;
    y += func(a) + func(b);
    for (int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }

    In = Integral(a, b, n, y);
    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 1" << endl;
    int n[5] = { 100, 1000, 10000, 100000, 1000000 };
```



```

a = 0.0;
b = 1.0;
int len_n = sizeof(n) / sizeof(int);

cout << "Линейное выполнение: " << endl;
for (int k = 0; k < len_n; k++)
{
    without_parall(a, b, n[k]);
}

cout << endl;
return 0;
}

```

VTune и Inspector для последовательной программы: Запуск **VTune Amplifier** и **Inspector XE** проводился по приведённому в лекции 3 алгоритму:

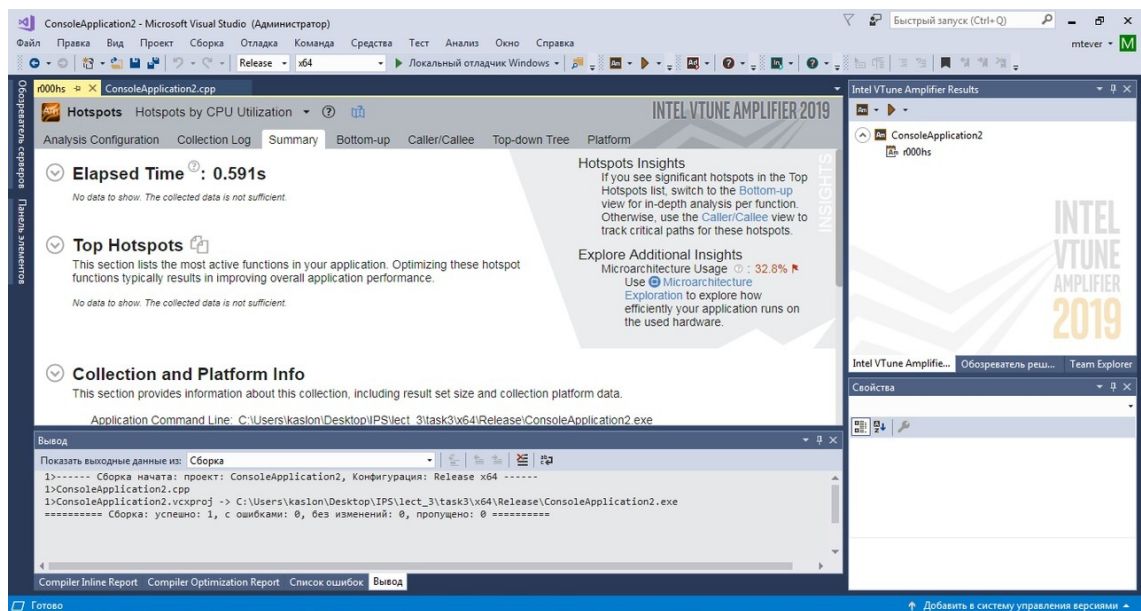


Рисунок 3. – Скриншот результатов работы VTune Amplifier

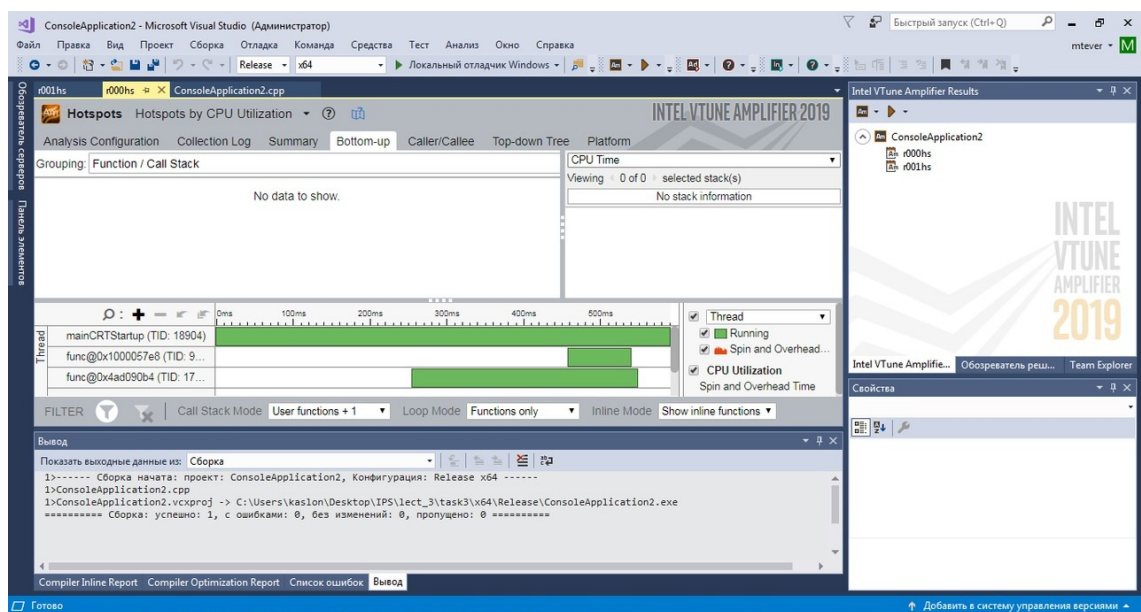


Рисунок 4. – Скриншот результатов работы VTune Amplifier

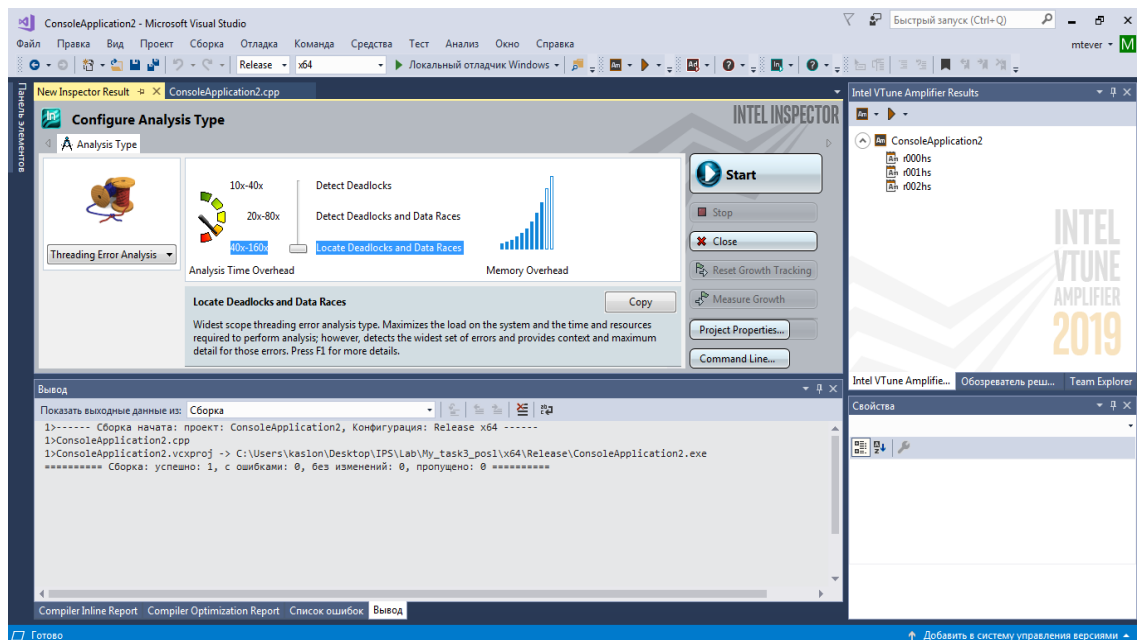


Рисунок 5. – Скриншот настроек Inspector XE

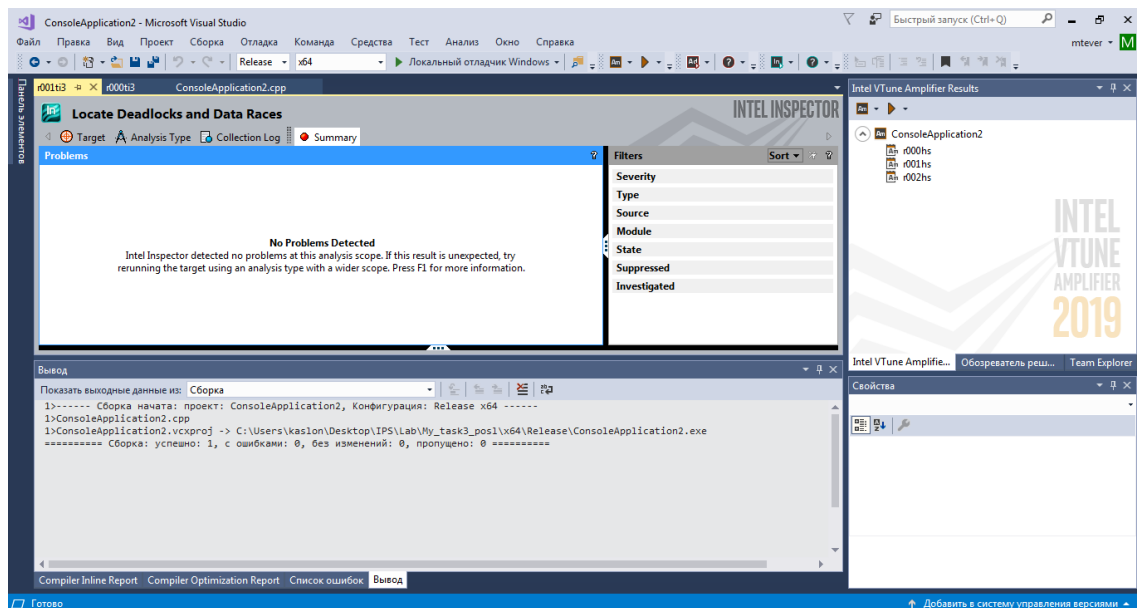


Рисунок 6. – Скриншот результатов работы Inspector XE

### Анализ полученных результатов:

Для последовательной программы анализаторы выдали:

1. VTune Amplifier – что данные не могут быть отображены
2. Inspector XE – Никакие проблемы не обнаружены

2) Добавим в программу `cilk_for`:

Код программы:

```
//Cilk Plus Reducers
#define HAS_CONDITIONAL_EXPLICIT 0
```

```

#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>

#include <math.h>
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>
// #include <cilk/reducer_opadd.h>

# define M_PI      3.14159265358979323846

using namespace std;
using namespace std::chrono;

double a, b, y, dy, In;
double timer_thread = 0;
double value_integral;

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}

double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}

void with_cilk_for(double a, double b, int n)
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    // cilk::reducer_opadd<long int> sum(0);
    // cilk::reducer_opadd<double> y(0);
    // y = 0;
    dy = (b - a) / n;
    y += func(a) + func(b);
    cilk_for(int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }

    In = Integral(a, b, n, y);
    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 3" << endl;
    int n[5] = { 100, 1000, 10000, 100000, 1000000 };

    a = 0.0;
    b = 1.0;
    int len_n = sizeof(n) / sizeof(int);

    cout << "\nВыполнение при помощи cilk_for: " << endl;
    for (int k = 0; k < len_n; k++)

```

```

{
    with_cilk_for(a, b, n[k]);
}

cout << endl;
return 0;
}

```

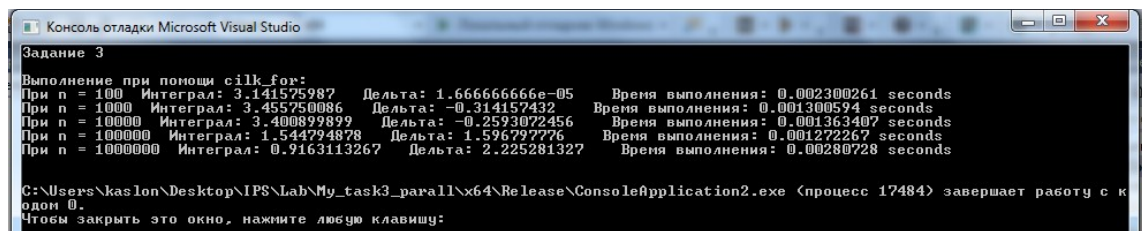


Рисунок 7. – Скриншот результатов вычислений интеграла по Заданию 3

Аналитическое решение	Необходимое количество разбиений	Численное решение	Дельта	Затраченное время, с
3.141592653589793	100	3.141575987	$1.(6) \cdot 10^{-5}$	$1.38 \cdot 10^{-3}$
	1000	3.455750086	-0.3141	$1.11 \cdot 10^{-3}$
	10000	3.084425223	0.05716	$1.22 \cdot 10^{-3}$
	100000	1.543522274	1.5981	$1.83 \cdot 10^{-3}$
	1000000	1.044514521	2.0971	$2.68 \cdot 10^{-3}$

Таблица 3. - Результаты вычисления интеграла по Заданию 3

### Анализ полученных результатов:

По полученных результатам наблюдается, что при увеличении количества разбиений идёт большее отклонение конечного результата.

Предположение – в коде программы необходимо использовать **reducer**.

### VTune и Inspector:

**VTune** и **Inspector** для программы с cilk\_for:

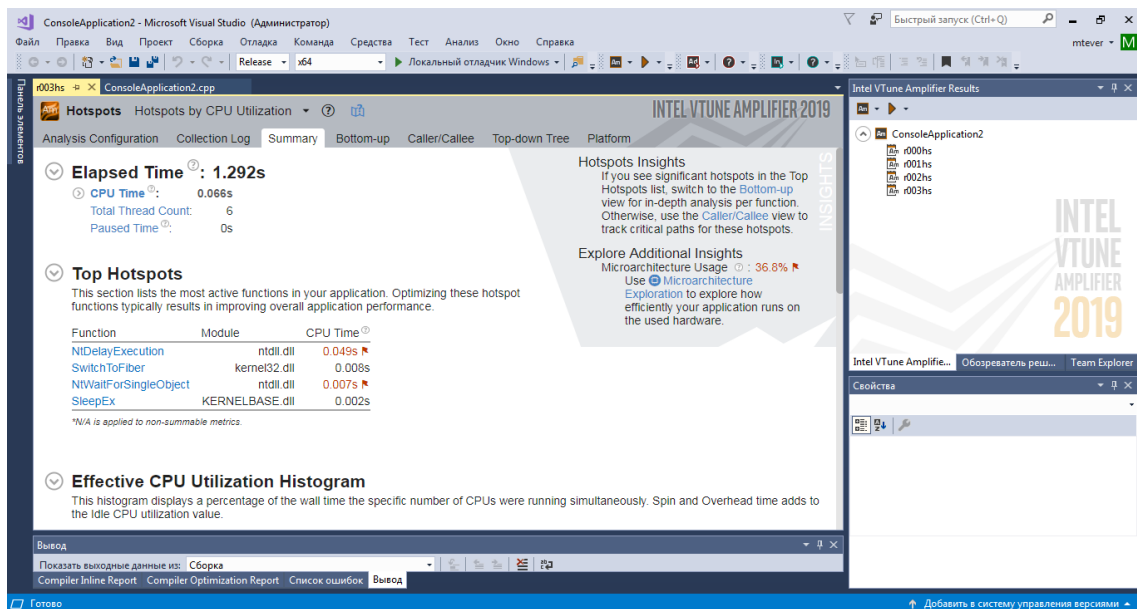


Рисунок 8. – Скриншот результатов работы VTune Amplifier

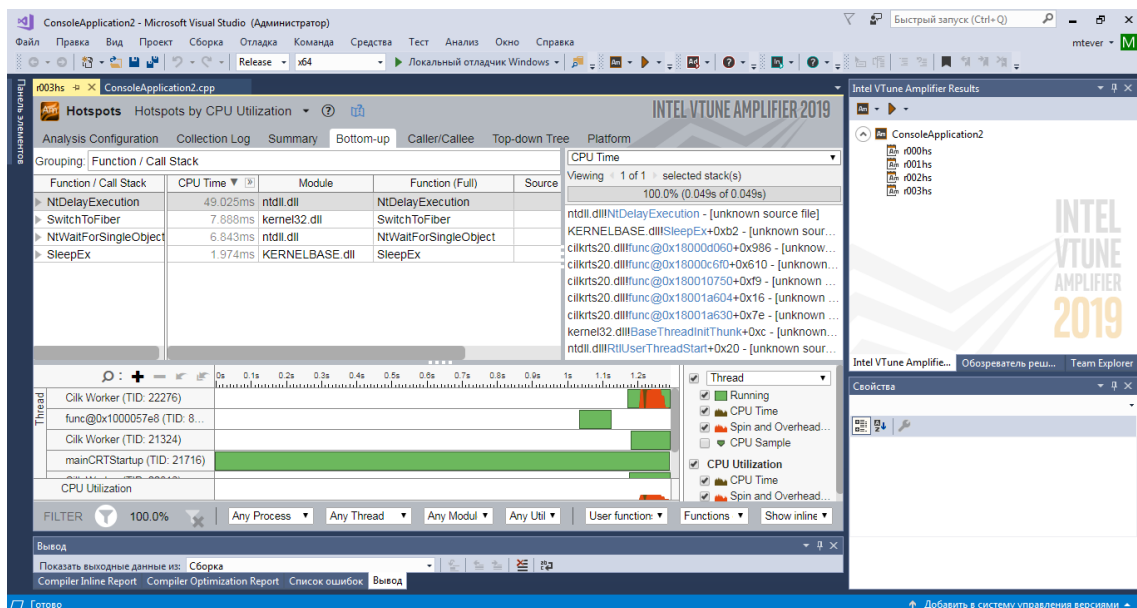


Рисунок 9. – Скриншот результатов работы VTune Amplifier

### Анализ полученных результатов:

Больше всего времени занимает функция NtDelayExecution. Если заглянем в информацию подробнее, то больше всего времени затрачено на функцию get (возврат из ближайшей процедуры)

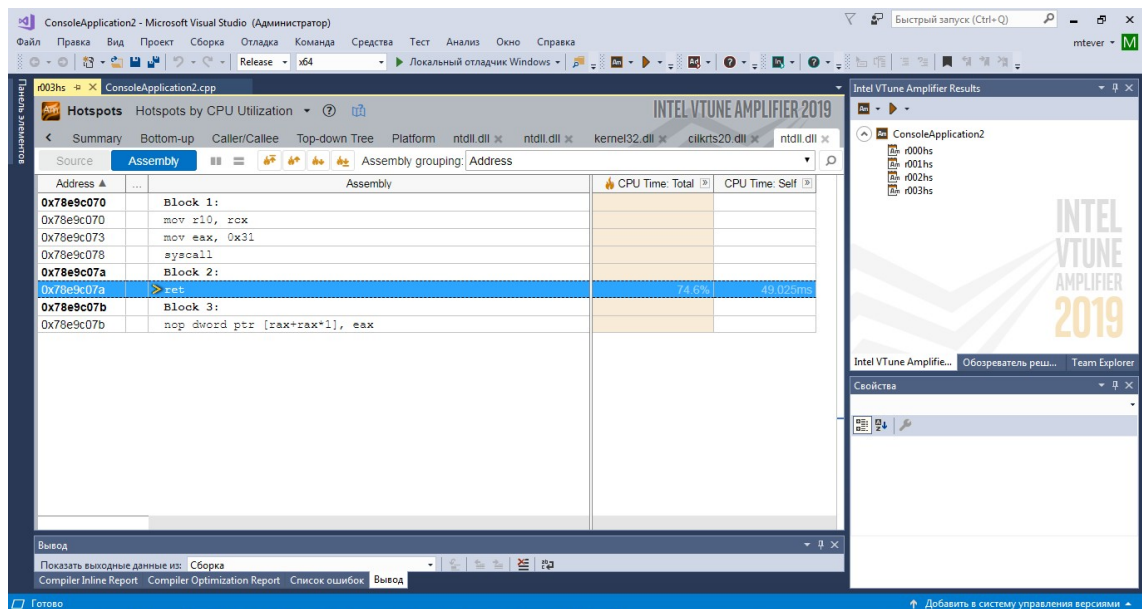


Рисунок 10. – Скриншот результатов работы VTune Amplifier

Результаты в **Inspector XE** на гонку данных и взаимной блокировке:

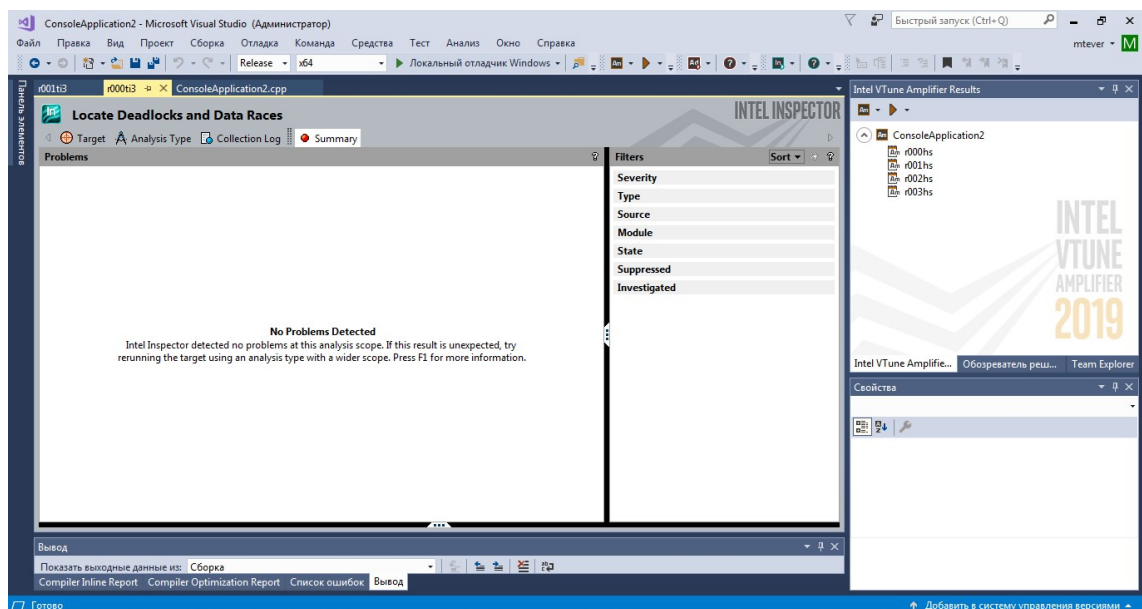


Рисунок 11. - Скриншот результатов работы Inspector XE

### Анализ полученных результатов:

Inspector XE пишет, что проблемы не обнаружены. Явно несостыковка.

Но при анализе Локальной памяти:



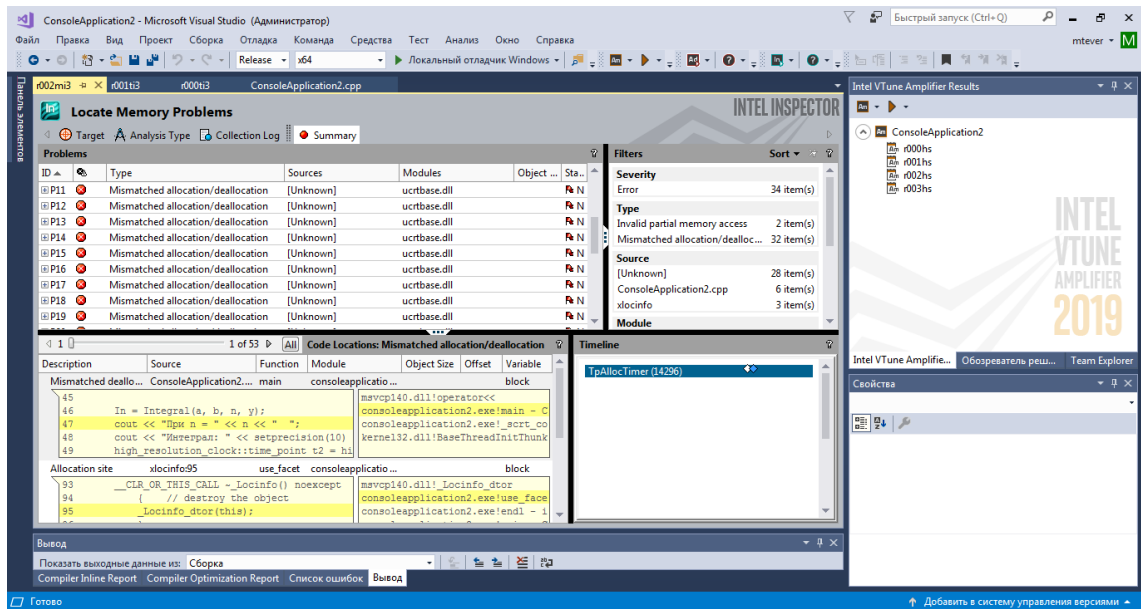


Рисунок 12. - Скриншот результатов работы Inspector XE

### Анализ полученных результатов:

Необходимо добавление reduce!

В данной программе необходимо сложение получаемых вычислений интеграла, используем reducer\_opadd.

### Код обновлённой программы:

```
//Cilk Plus Reducers
#define _HAS_CONDITIONAL_EXPLICIT 0
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>

#include <math.h>
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>
//#include <cilk/reducer_opadd.h>

# define M_PI      3.14159265358979323846

using namespace std;
using namespace std::chrono;

double a, b, y, dy, In;
double timer_thread = 0;
double value_integral;

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}
double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}
```

```

void with_cilk_for(double a, double b, int n)
{
    double sum = 0;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    //cilk::reducer_opadd<long int> sum(0);
    //cilk::reducer_opadd<double> y(0);
    y = 0;
    dy = (b - a) / n;
    // y += func(a) + func(b);
    cilk::reducer_opadd <double> y(func(a) + func(b));
    cilk_for(int ii = 1; ii < n; ii++)
    {
        y += 2 * (func(a + dy * ii));
    }
    sum = y.get_value();
    In = Integrāl(a, b, n, sum);
    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 3" << endl;
    int n[5] = { 100, 1000, 10000, 100000, 1000000 };

    a = 0.0;
    b = 1.0;
    int len_n = sizeof(n) / sizeof(int);

    cout << "\nВыполнение при помощи cilk_for: " << endl;
    for (int k = 0; k < len_n; k++)
    {
        with_cilk_for(a, b, n[k]);
    }

    cout << endl;
    return 0;
}

```

Добавленные строки, помимо библиотек `#include <cilk/reducer_opadd.h>`:

```

cilk::reducer_opadd <double> y(func(a) + func(b));
cilk_for(int ii = 1; ii < n; ii++)
{
    y += 2 * (func(a + dy * ii));
}
sum = y.get_value();

```

Результат выполнения программы:



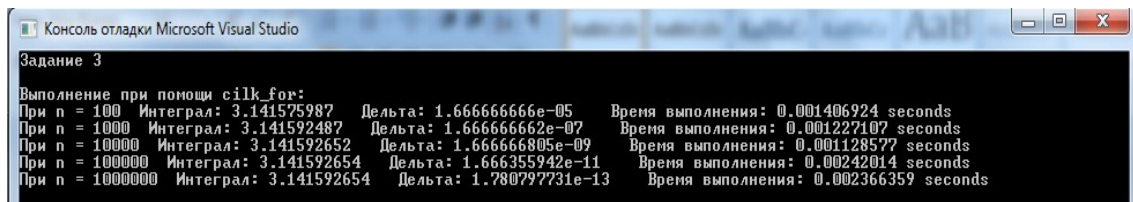


Рисунок 13. – Скриншот результатов вычислений интеграла по Заданию 3

Аналитическое решение	Необходимое количество разбиений	Численное решение	Дельта	Затраченное время, с
3.141592653589793	100	3.141575987	$1.(6) \cdot 10^{-5}$	$1.41 \cdot 10^{-3}$
	1000	3.141592487	$1.(6) \cdot 10^{-7}$	$1.23 \cdot 10^{-3}$
	10000	3.141592652	$1.(6) \cdot 10^{-9}$	$1.11 \cdot 10^{-3}$
	100000	3.141592654	$1.(6) \cdot 10^{-11}$	$2.42 \cdot 10^{-3}$
	1000000	3.141592654	$1.78 \cdot 10^{-13}$	$2.36 \cdot 10^{-3}$

Таблица 3. - Результаты вычисления интеграла по Заданию 3

### Анализ полученных результатов:

Действительно, добавление **reducer\_opadd** исправило ошибку в результате. С увеличением количества разбиений мы также можем наблюдать приближение результата к аналитическому решению.

Оценим данную программу в **VTune** и **Inspector**:

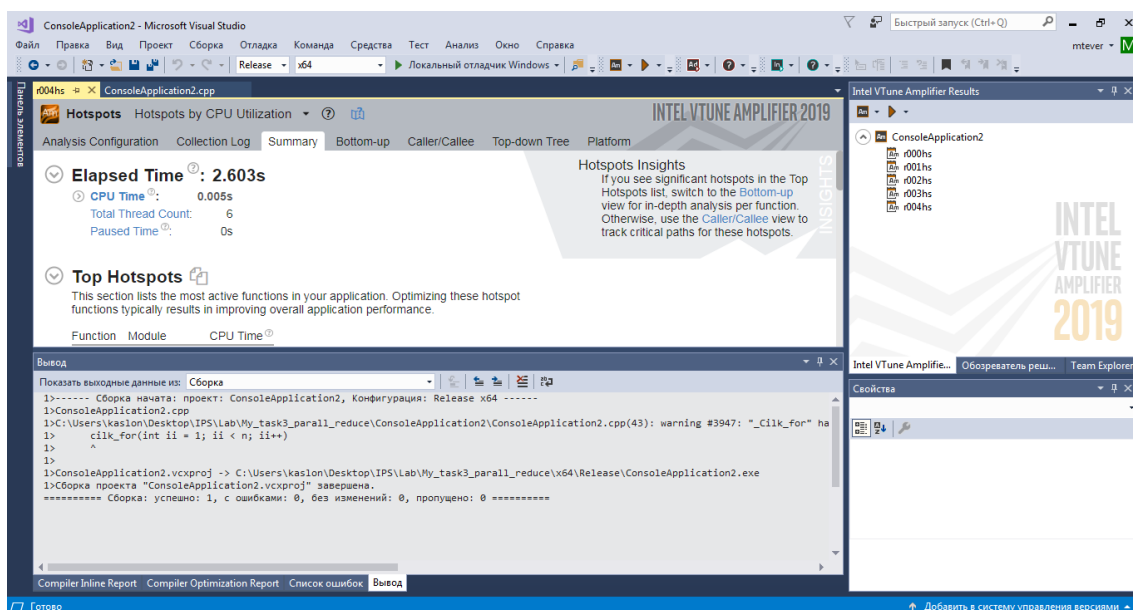


Рисунок 14. – Скриншот результатов анализа VTune Amplifier

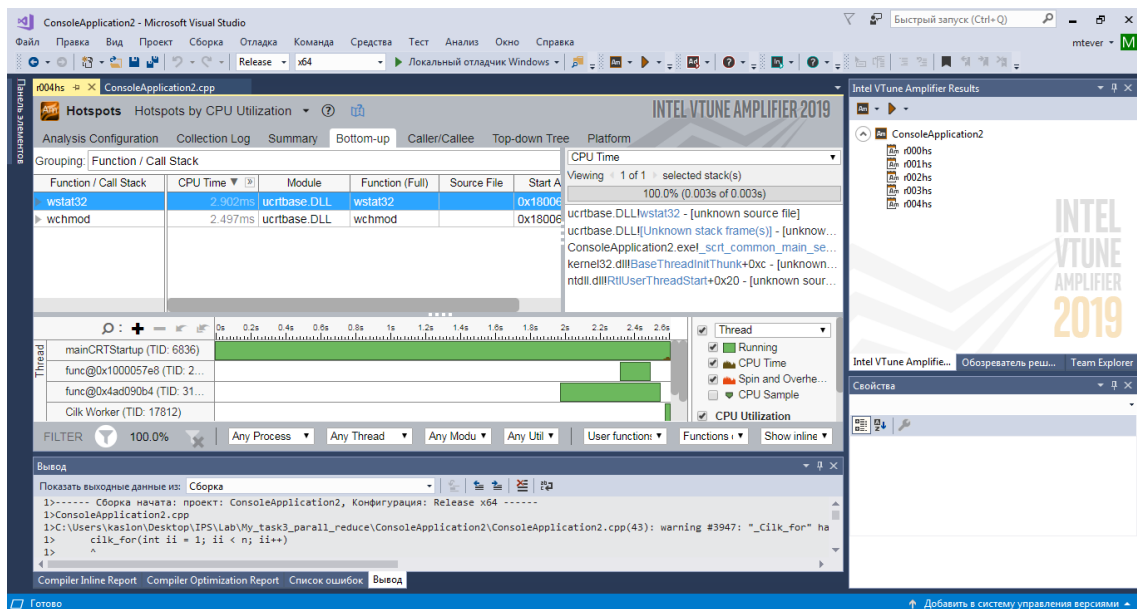


Рисунок 15. – Скриншот результатов анализа VTune Amplifier

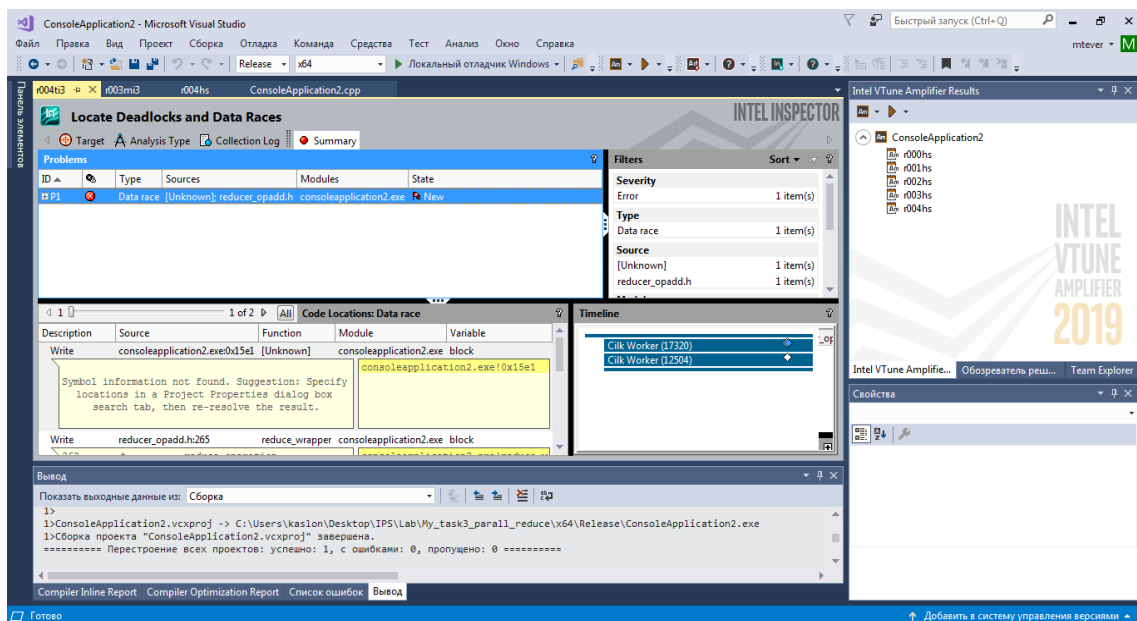


Рисунок 15. – Скриншот результатов анализа Inspector XE

### Анализ полученных результатов:

- 1) Численные результаты соответствуют аналитическому решению.
- 2) По анализу **VTune Amplifier** видно, что больше функция get не занимает огромное количество времени выполнения программы
- 3) По анализу Inspector XE: reducer\_opadd указывает на гонку данных

#### Задание 4: Программа с использованием шаблонов TBB

Код программы:

```
#define _HAS_CONDITIONAL_EXPLICIT 0
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>

#include <math.h>
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>
#include <tbb/tbb.h>
#include <mutex>

# define M_PI      3.14159265358979323846

using namespace std;
using namespace std::chrono;

double a, b, y, dy, In;
double timer_thread = 0;
double value_integral;

mutex gmutex;

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}

double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}

void compute(double a, double b, int n)
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    dy = (b - a) / n;
    y += func(a) + func(b);
    tbb::parallel_for(int(0), n, [&](size_t ii)
    {
        gmutex.lock();
        y += 2 * (func(a + dy * ii));
        gmutex.unlock();
    });
    In = Integral(a, b, n, y);

    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 4" << endl;
    int n[5] = { 100, 1000, 10000, 100000, 1000000 };
```

```

a = 0.0;
b = 1.0;
int len_n = sizeof(n) / sizeof(int);

cout << "\nВыполнение при помощи TBB: " << endl;
for (int k = 0; k < len_n; k++)
{
    compute(a, b, n[k]);
}

cout << endl;
return 0;
}

```

Помимо добавления библиотеки `#include <tbb/tbb.h>` я добавил `#include <mutex>`  
В коде были исправлены участки:

1) Цикл:

```

tbb::parallel_for(int(0), n, [&](size_t ii)
{
    gmutex.lock();
    y += 2 * (func(a + dy * ii));
    gmutex.unlock();
});

```

2) Объявление

```
mutex gmutex;
```

Результат:

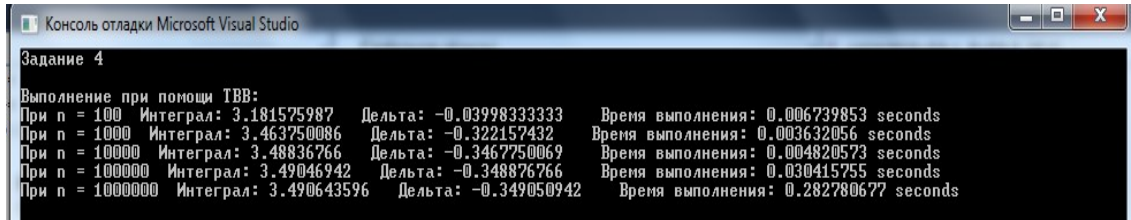


Рисунок 16. – Результат при использовании шаблонов TBB

### Анализ полученных результатов:

- 1) Без использования `mutex` получаемый результат «плавал».
- 2) Полученные результаты отличаются от аналитического решения, но результат ближе получить не удалось без `reducer`.

**Код программы с использованием `reducer`:** Добавлена конструкция

```

cilk::reducer_opadd <double> y(func(a) + func(b));

double res = y.get_value();

```

Весь код:

```

#define _HAS_CONDITIONAL_EXPLICIT 0
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>
#include <chrono>

#include <math.h>
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>

```

```

#include <tbb/tbb.h>
#include <mutex>

# define M_PI      3.14159265358979323846

using namespace std;
using namespace std::chrono;

double a, b, y, dy, In;
double timer_thread = 0;
double value_integral;

mutex gmutex;

double func(double x)
{
    return 4 / (1 + pow(x, 2));
}

double Integral(double a, double b, int n, double y)
{
    return ((b - a) / (2 * n) * y);
}

void compute(double a, double b, int n)
{
    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    dy = (b - a) / n;
    //y += func(a) + func(b);
    cilk::reducer_opadd <double> y(func(a) + func(b));
    tbb::parallel_for(int(0), n, [&](size_t ii)
    {
        gmutex.lock();
        y += 2 * (func(a + dy * ii));
        gmutex.unlock();
    });
    double res = y.get_value();
    In = Integral(a, b, n, res);

    cout << "При n = " << n << " ";
    cout << "Интеграл: " << setprecision(10) << In << " ";
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    duration<double> duration = (t2 - t1);
    cout << "Дельта: " << M_PI - In << " ";
    cout << "Время выполнения: " << duration.count() << " seconds" << endl;
}

int main()
{
    system("mode con cols=120 lines=50");
    setlocale(LC_ALL, "Rus");
    cout << "Задание 4" << endl;
    int n[5] = { 100, 1000, 10000, 100000, 1000000 };

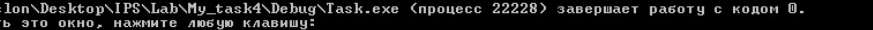
    a = 0.0;
    b = 1.0;
    int len_n = sizeof(n) / sizeof(int);

    cout << "\nВыполнение при помощи TBB: " << endl;
    for (int k = 0; k < len_n; k++)
    {
        compute(a, b, n[k]);
    }

    cout << endl;
    return 0;
}

```

### Результат с использованием reducer:



Консоль отладки Microsoft Visual Studio

Задание 4

Выполнение при помощи TBB:

n	Интеграл	Дельта	Время выполнения
100	3.181575987	-0.03998333333	0.010789019 seconds
1000	3.145592487	-0.003998333333	0.002821239 seconds
10000	3.141992652	-0.0003999833333	0.006522677 seconds
100000	3.141632654	-3.99998337e-05	0.046487205 seconds
1000000	3.141596654	-3.9999985e-06	0.398122185 seconds

C:\Users\kasion\Desktop\IPS\Lab\My\_task4\Debug\Task.exe (процесс 22228) завершает работу с кодом 0.  
Чтобы закрыть это окно, нажмите любую клавишу:

Рисунок 17. – Результат при использовании шаблонов ТВВ

<i>Аналитическое решение</i>	<i>Необходимое количество разбиений</i>	<i>Численное решение</i>	<i>Дельта</i>	<i>Затраченное время, с</i>
3.141592653589793	100	3.181575987	$3.(9)*10^{-2}$	$1.08*10^{-2}$
	1000	3.145592487	$3.(9)*10^{-3}$	$2.82*10^{-3}$
	10000	3.141992652	$3.(9)*10^{-4}$	$6.52*10^{-3}$
	100000	3.141632654	$3.(9)*10^{-5}$	$4.46*10^{-2}$
	1000000	3.141596954	$3.(9)*10^{-6}$	$3.98*10^{-1}$

Таблица 5. – Результаты Задания 4