

Индивидуальное задание

Напишите параллельную программу вычисления следующего интеграла с использованием дополнений **Intel Cilk Plus** языка C++:

$$\int_0^1 \frac{4}{1+x^2} dx$$

Возьмем интеграл аналитически:

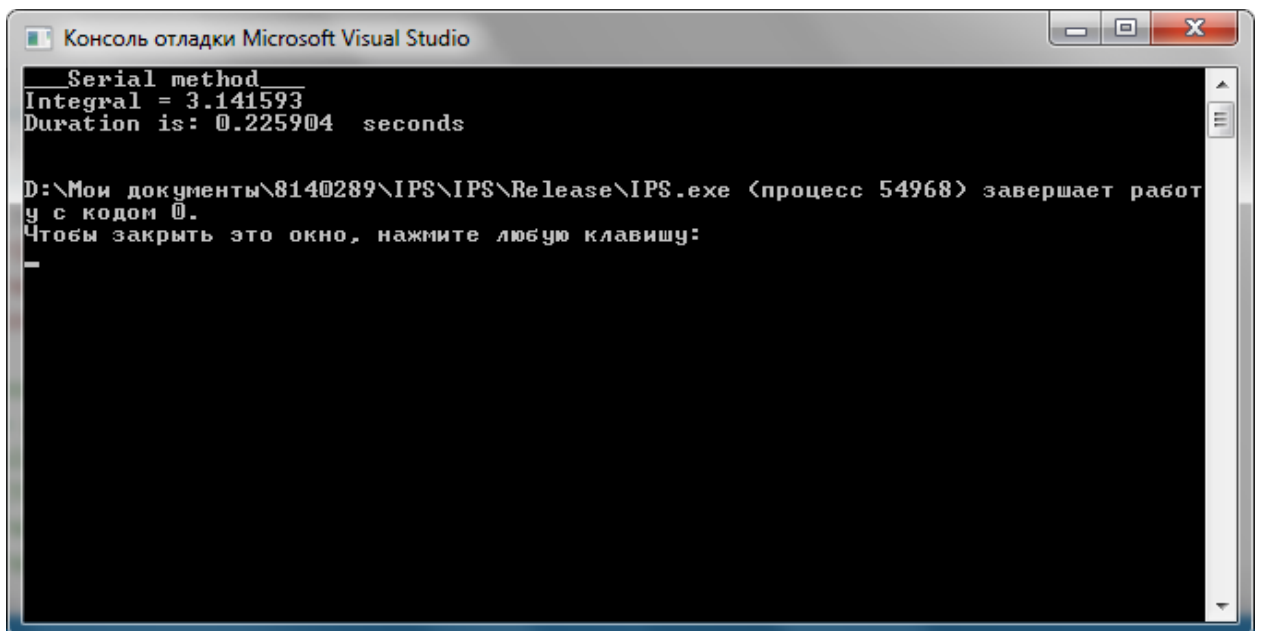
Мы знаем, что $\arctg(x)' = \frac{1}{1+x^2}$. Тогда используя формулу Ньютона-Лейбница получим:

$$\int_0^1 \frac{4}{1+x^2} dx = 4 * (\arctg(1) - \arctg(0)) = 4 * \left(\frac{\pi}{4} - 0\right) = \pi$$

Далее будем находить значение интеграла численно, например, можно воспользоваться простым методом прямоугольников.

Реализуем последовательное вычисление интеграла:

```
// Последовательная реализация метода прямоугольников
double serialFindIntegral(int n)
{
    double h = (b - a)/(n - 1);
    double S = 0.0;
    for (size_t i = 0; i < n; i++) {
        S += fun(i*h) + fun((i + 1)*h);
    }
    return S*h/2.0;
}
```



Консоль отладки Microsoft Visual Studio

```
Serial method
Integral = 3.141593
Duration is: 0.225904 seconds

D:\Мои документы\8140289\IPS\IPS\Release\IPS.exe (процесс 54968) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
-
```

Метод работает верно.

Обзор инструментов параллелизации:

- Intel Cilk Plus — расширение языка Си++, призванное упростить написание многопоточных программ. Cilk Plus представляет собой динамический планировщик исполнения потоков и набор ключевых слов, сообщающих компилятору о возможности применения той или иной схемы планирования.
- Intel Parallel Inspector — инструмент для обнаружения ошибок памяти и потоков в последовательных и параллельных приложениях на платформах Windows и Linux.
- Intel VTune Amplifier — это средство для оптимизации производительности и профилировки параллельных приложений.

Определим наиболее часто используемые участки кода с помощью **Intel VTune Amplifier XE**.

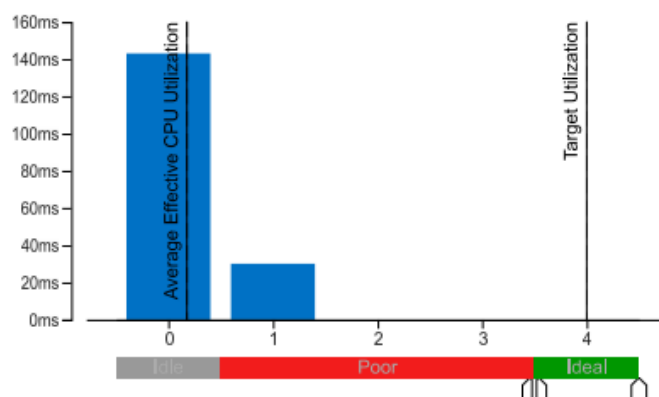
Hotspots Hotspots by CPU Utilization ?

Analysis Configuration	Collection Log	Summary	Bottom-up	Caller/Callee	Top-down Tree	Platform
Function	Module	CPU time				
main	lab3.exe	0.030s				
func@0x180052100	ucrtbase.dll	0.001s				
Query_perf_frequency	MSVCP140.dll	0.000s				

*N/A is applied to non-summable metrics.

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.



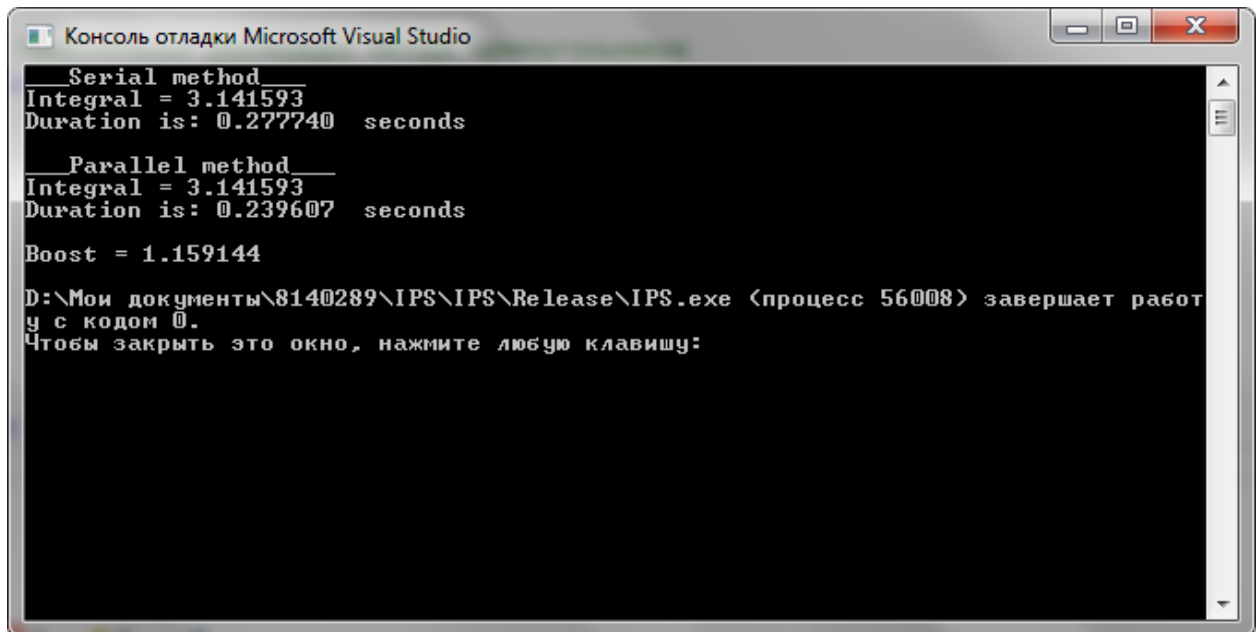
Реализуем параллельный метод прямоугольников:

```
// Параллельная реализация метода прямоугольников
double parallelFindIntegral(int n)
{
    double h = (b - a)/n;
    cilk::reducer_opadd<double> S(0.0);
```

```

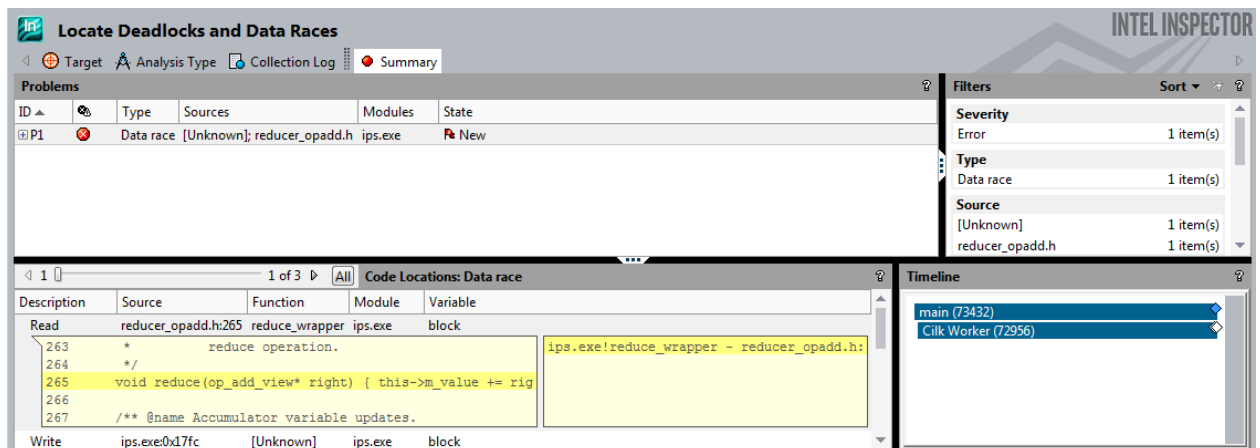
    cilk_for(size_t i = 0; i < n; i++) {
        S += fun(i*h) + fun((i + 1)*h);
    }
    return S->get_value()*h/2.0;
}

```



Видно, что параллельный метод также дает верный результат. Также можно наблюдать небольшое ускорение.

Далее проверим программу с помощью инструмента **Intel Parallel Inspector XE**.



Ошибок не обнаружено.

Далее оценим зависимость времени выполнения от заданных параметров алгоритма.

```

// Зависимость времени выполнения от числа итераций N
int timeCompare()
{
    double ans;
    high_resolution_clock::time_point ts1, ts2, tp1, tp2;

```

```

printf("__Serial method__|__Parallel method__ \n\n");
for (size_t i = 100; i < N*1000; i *= 10)
{
    ts1 = high_resolution_clock::now();
    ans = serialFindIntegral(N);
    ts2 = high_resolution_clock::now();
    duration_s = (ts2 - ts1);

    tp1 = high_resolution_clock::now();
    ans = serialFindIntegral(N);
    tp2 = high_resolution_clock::now();
    duration_p = (tp2 - tp1);
    printf("      %1f      |      %1f \n", duration_s.count(),
duration_p.count());
}
return 0;
}

```

```

__Serial method__|__Parallel method__
0.000003      | 0.000003
0.000003      | 0.000002
0.000003      | 0.000002
0.000003      | 0.000002
0.000003      | 0.000003
0.000003      | 0.000003
0.000003      | 0.000002
0.000003      | 0.000002
D:\Мои документы\8140289\IPS\IPS\Release\IPS.exe (процесс 100460) завершает рабо
ту с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:

```

Местами на больших значениях N наблюдается более быстрая работа параллельного метода.