**Лабораторная работа №1**

**Колбин Павел**

**Вариант 10**

1. 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).

Функция последовательного вычисления интеграла

|  |
| --- |
| void computeIntegral(double eps) {  double a = -1, b = 1, computedInt = 5.141592653589793;  double I = eps + 1, I1 = 0;  high\_resolution\_clock::time\_point start = high\_resolution\_clock::now();  for (int N = 2; (N <= 4) || (fabs(I1 - I) > eps); N \*= 2)  {  double h, sum2 = 0, sum4 = 0, sum = 0;  h = (b - a) / (2 \* N);  for (int i = 1; i <= 2 \* N - 1; i += 2)  {  sum4 += f(a + h \* i);  sum2 += f(a + h \* (i + 1));  }  sum = f(a) + 4 \* sum4 + 2 \* sum2 - f(b);  I = I1;  I1 = (h / 3) \* sum;  }  high\_resolution\_clock::time\_point finish = high\_resolution\_clock::now();  duration<double> duration = (finish - start);  cout << "Duration is: " << duration.count() << " seconds" << endl;  cout << "EPS is: " << setprecision(10) << abs(computedInt - I1) << endl;  } |

Таблица 1 - данные получившиеся при последовательном решении

|  |  |  |  |
| --- | --- | --- | --- |
| *Аналитическое решение* | *Необходимое количество разбиений* | *Epsilon* | *Затраченное время, c* |
| 5.141592653589793 | *100* | *1.686039744e-05* | *7.5e-06* |
| *1000* | *1.686039744e-05* | *6.8e-06* |
| *10000* | *1.02663045e-06* | *5.3e-06* |
| *100000* | *6.37257962e-08* | *1.04e-05* |
| *1000000* | *6.37257962e-08* | *1.17e-05* |
| *10000000* | *3.975952012e-09* | *2.43e-05* |

2. 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?

Таблица 2 - данные получившиеся при использовании #pragma loop(hint\_parallel(8))

|  |  |  |  |
| --- | --- | --- | --- |
| *Аналитическое решение* | *Необходимое количество разбиений* | *Epsilon* | *Затраченное время, c* |
| 5.141592653589793 | *100* | *1.686039744e-05* | *2.8e-06* |
| *1000* | *1.686039744e-05* | *5.4e-06* |
| *10000* | *1.02663045e-06* | *7.2e-06* |
| *100000* | *6.37257962e-08* | *1.1e-05* |
| *1000000* | *6.37257962e-08* | *1.56e-05* |
| *10000000* | *3.975952012e-09* | *2.18e-05* |

Как можно увидеть выйгрыша по времени выполнения практически нет.

Таблица 3 - данные получившиеся при ручном создании потоков

|  |  |  |  |
| --- | --- | --- | --- |
| *Аналитическое решение* | *Необходимое количество разбиений* | *Epsilon* | *Затраченное время, c* |
| 5.141592653589793 | *100* | *6.37257962e-08* | *0.0892074* |
| *1000* | *6.37257962e-08* | *0.0208014* |
| *10000* | *6.37257962e-08* | *0.0146707* |
| *100000* | *6.37257962e-08* | *0.0155395* |
| *1000000* | *7.771636668e-08* | *0.0130845* |
| *10000000* | *4.869564307e-09* | *0.0161003* |

А вот тут можно увидеть, что время, затраченное на выполнение программы с помощью ручного создания потоков на порядки, возросло, потому что программная процедура создания и удаления потока, довольно затратная по времени.

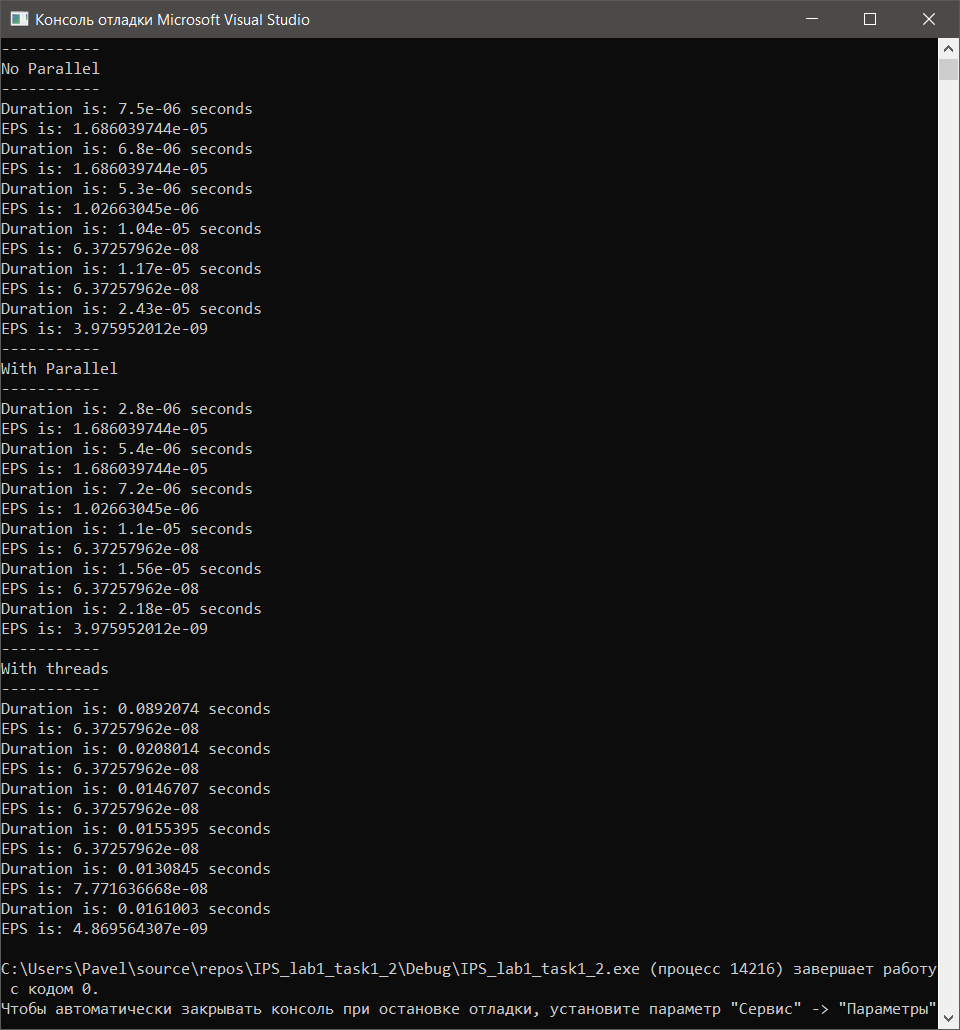


Рисунок 1 - пример выполнения программы

3. 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.

Пример запуска VTune и Inspector для последовательной программы.

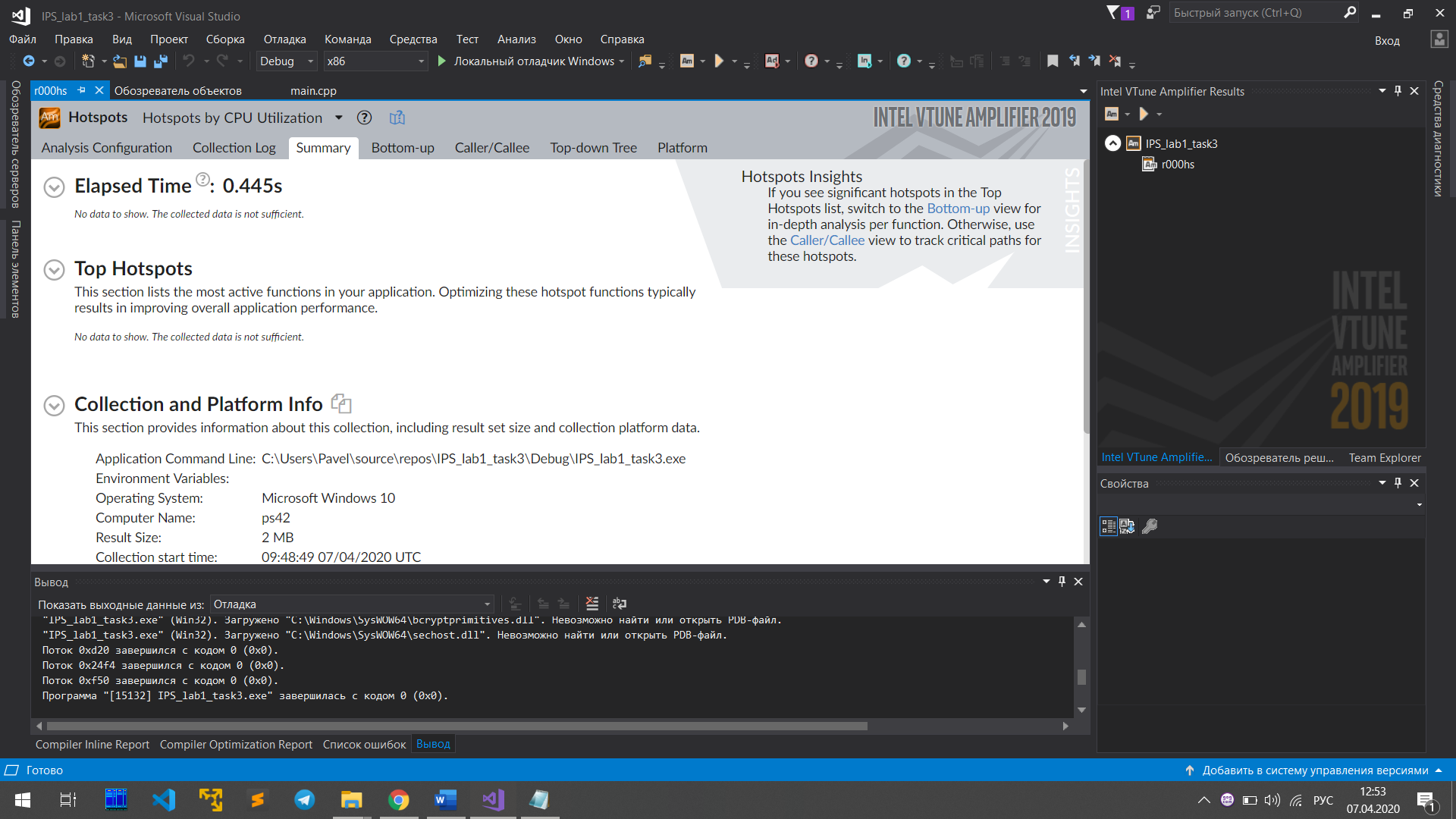


Рисунок 2 – пример работы VTune для последовательной программы

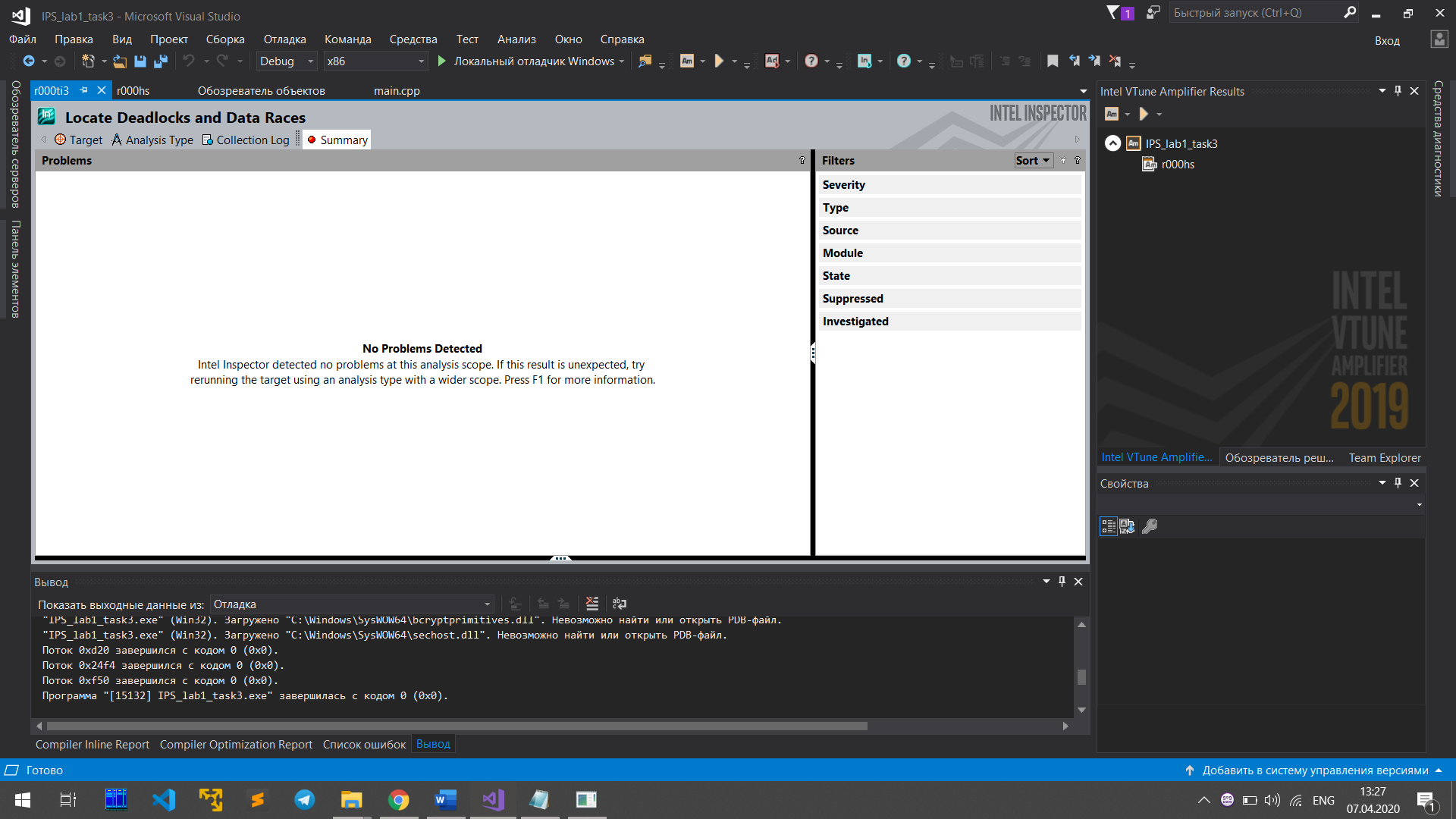


Рисунок 3 – пример работы Inspector на наличие гонок данных для последовательной программы

Добавим в программу cilk\_for и запустим VTune и Inspector

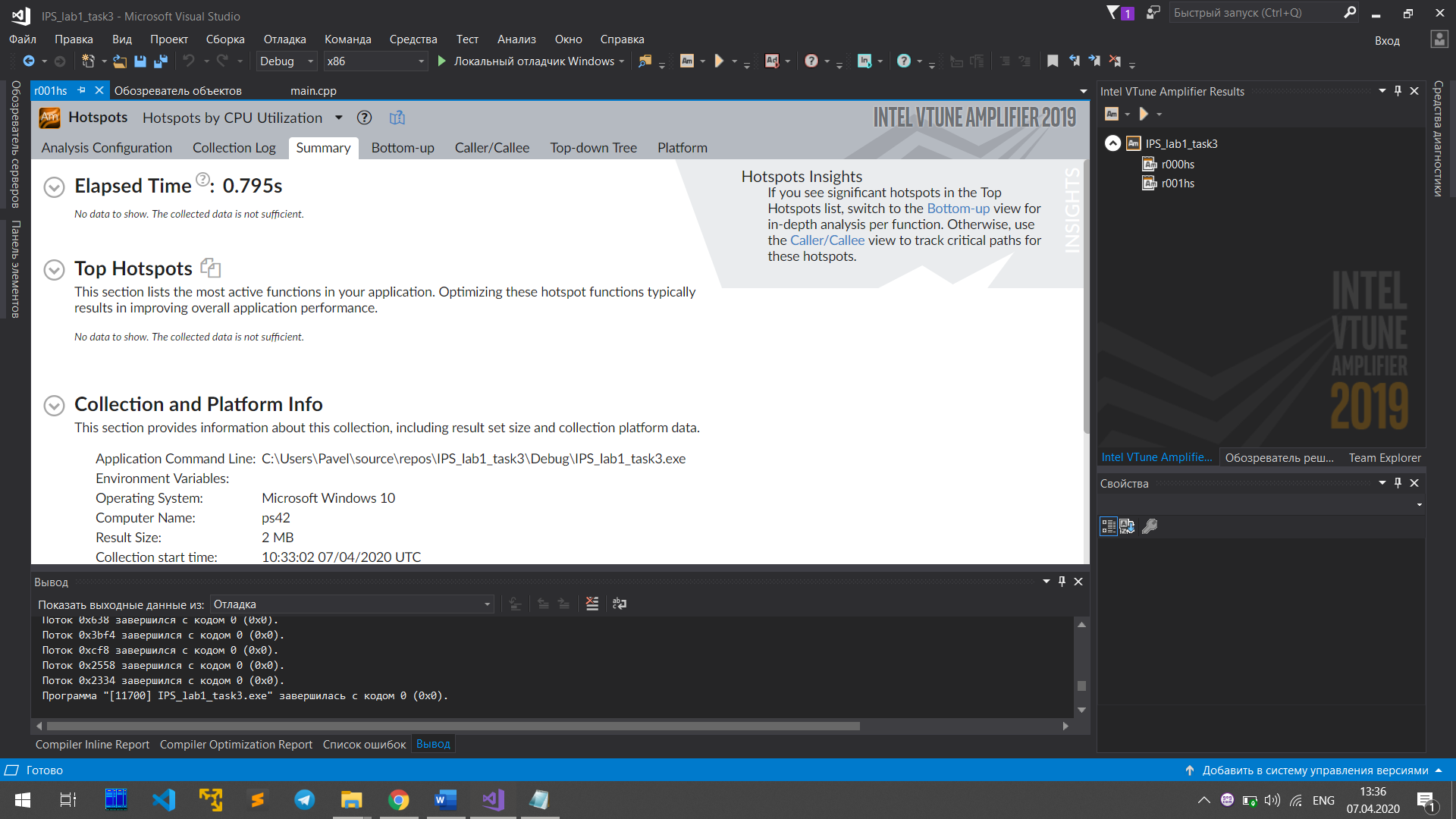


Рисунок 4 - пример работы VTune для программы с cilk\_for

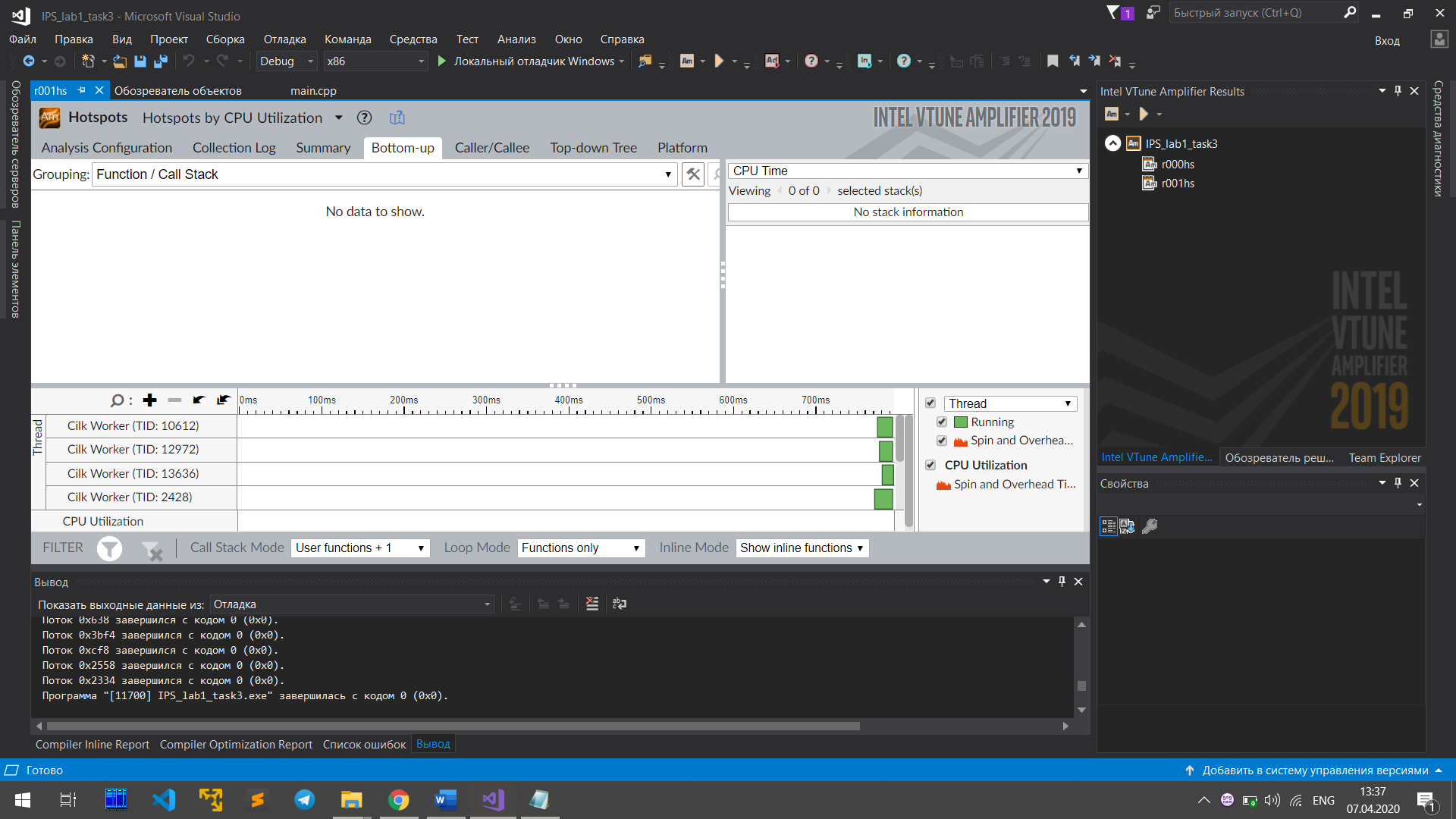


Рисунок 5 - пример работы VTune для программы с cilk\_for

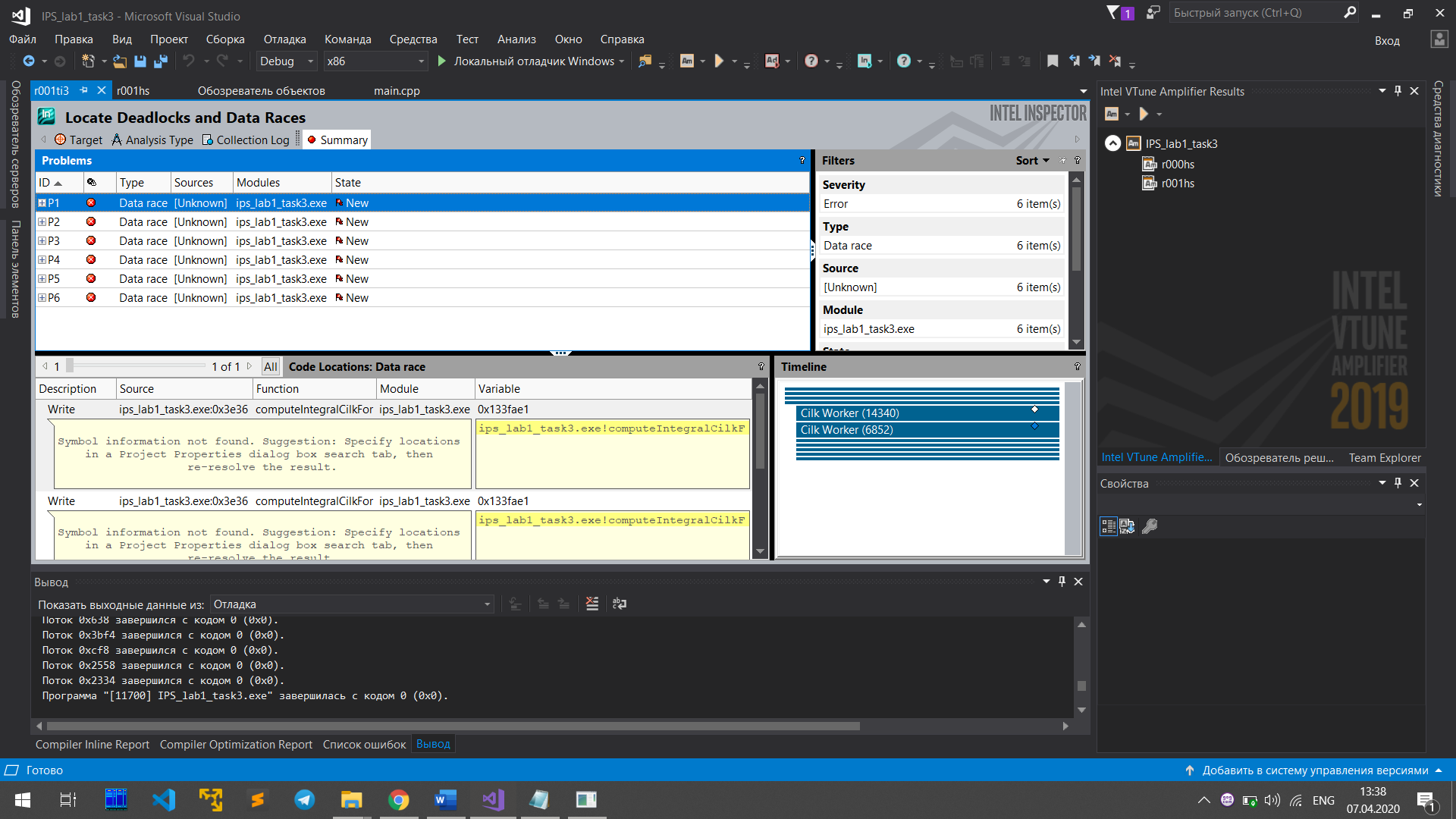


Рисунок 6 – пример работы Inspector на наличие гонок данных для программы с cilk\_for

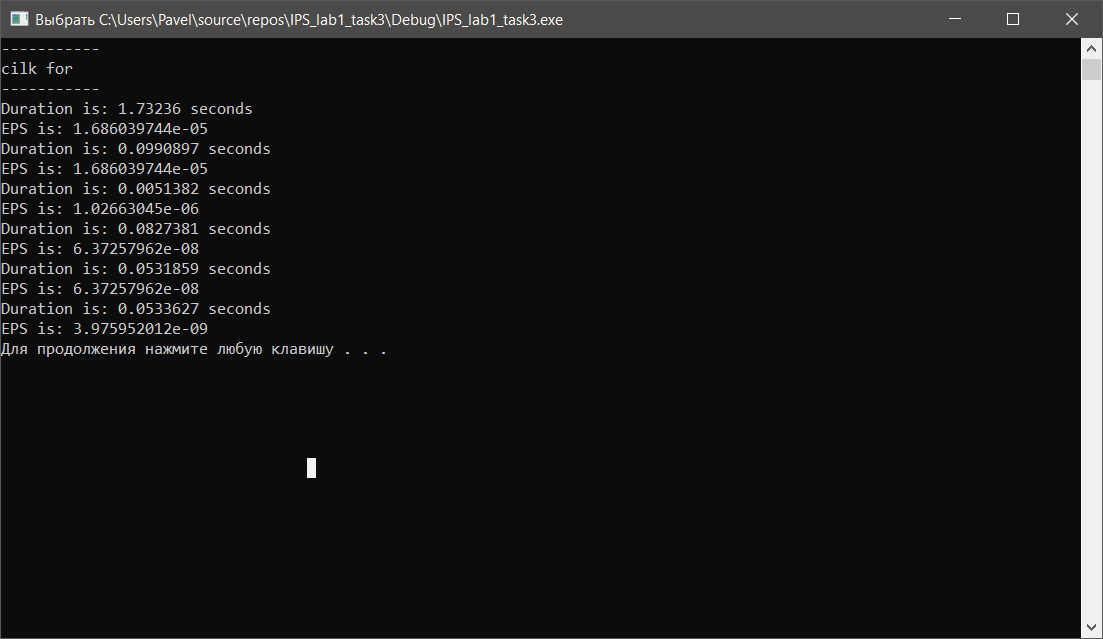


Рисунок 7 – результаты работ программы с cilk\_for

При добавлении cilk\_for получили ситуацию, при которой возникли гонки данных, необходимо добавтить reducer.

Добавим в программу reducer\_oppad и посмотрим на полученные результаты

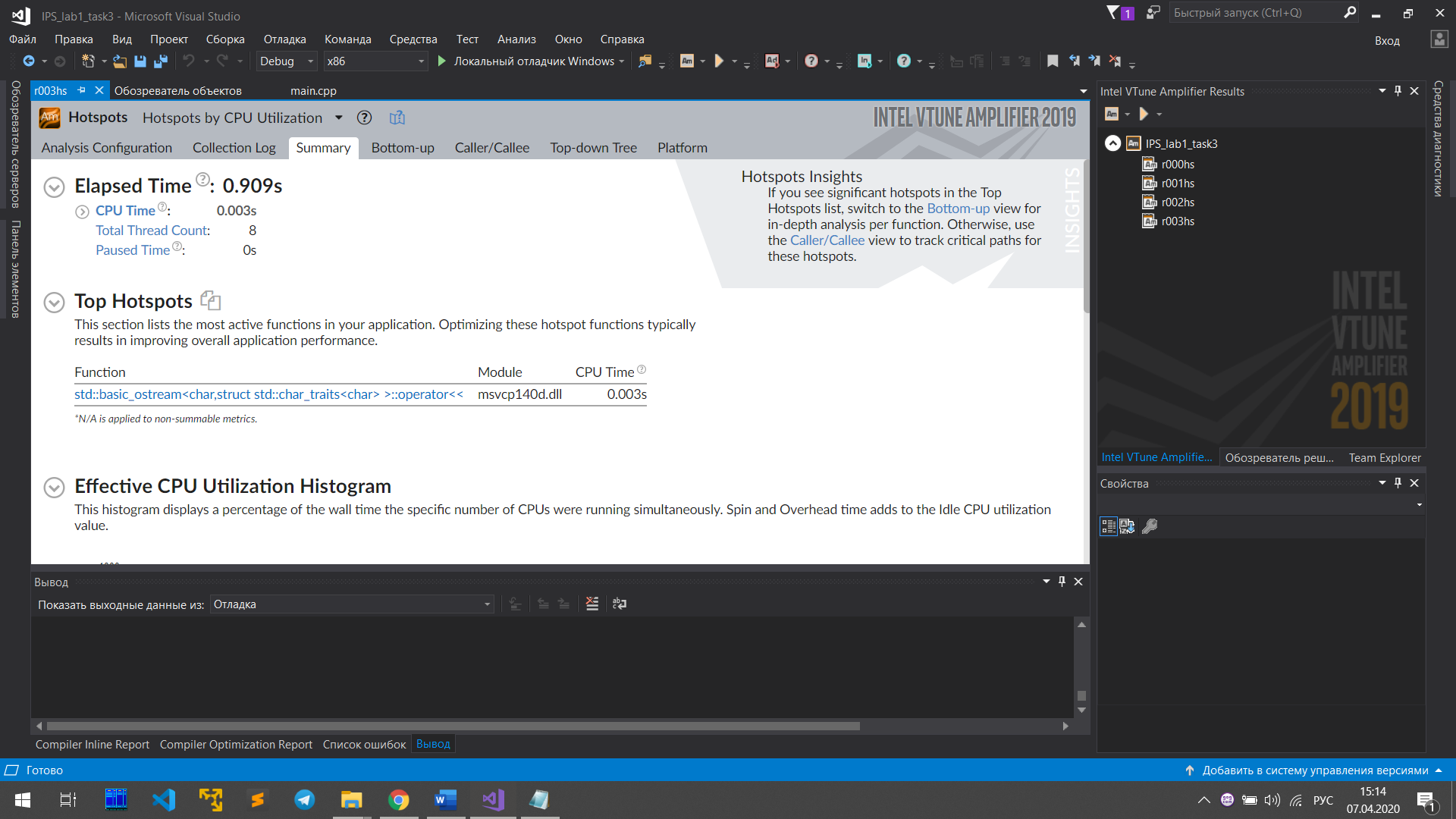


Рисунок 8 - пример работы VTune для программы с cilk\_for и reducer\_opadd

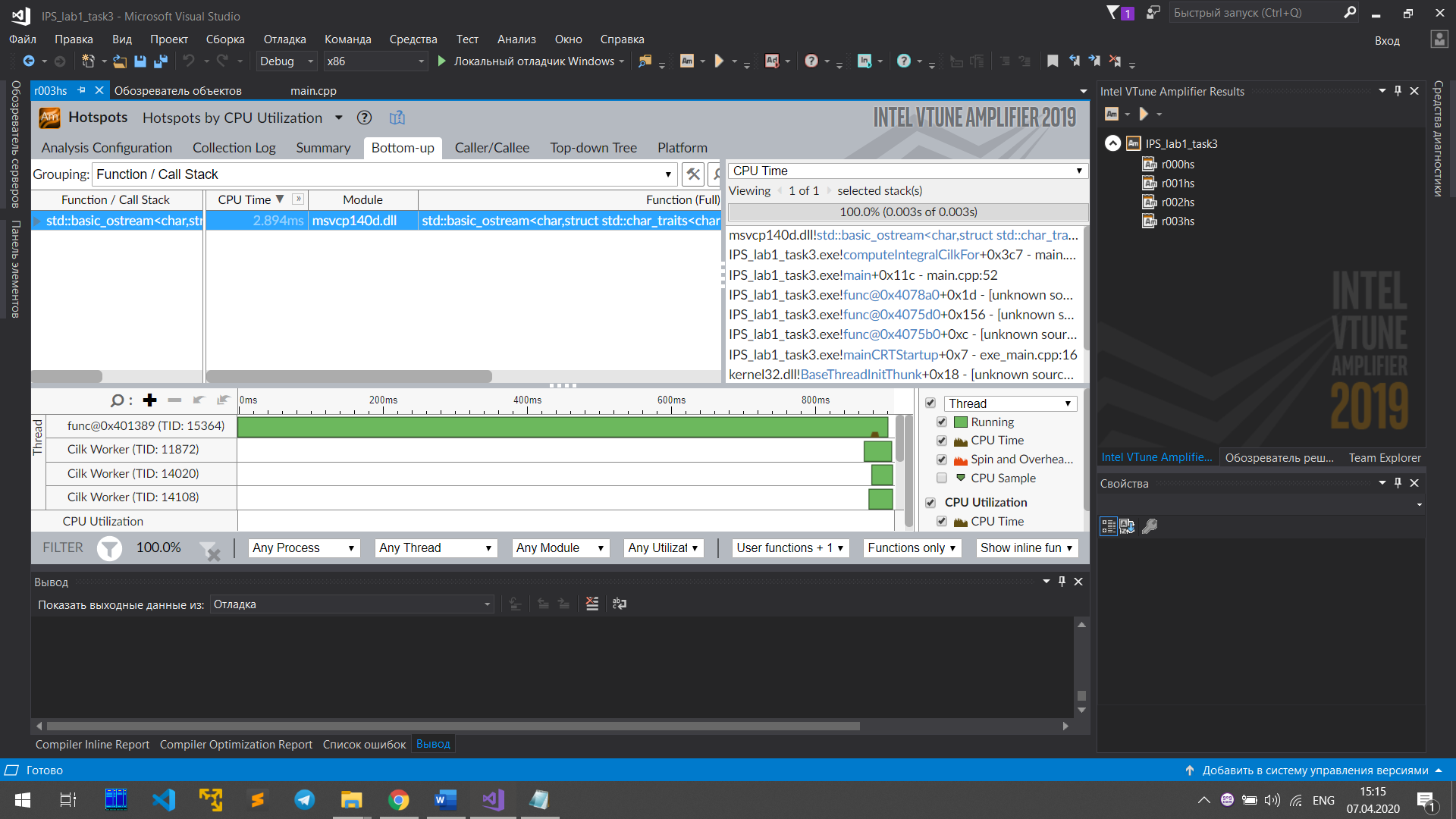


Рисунок 9 - пример работы VTune для программы с cilk\_for и reducer\_opadd

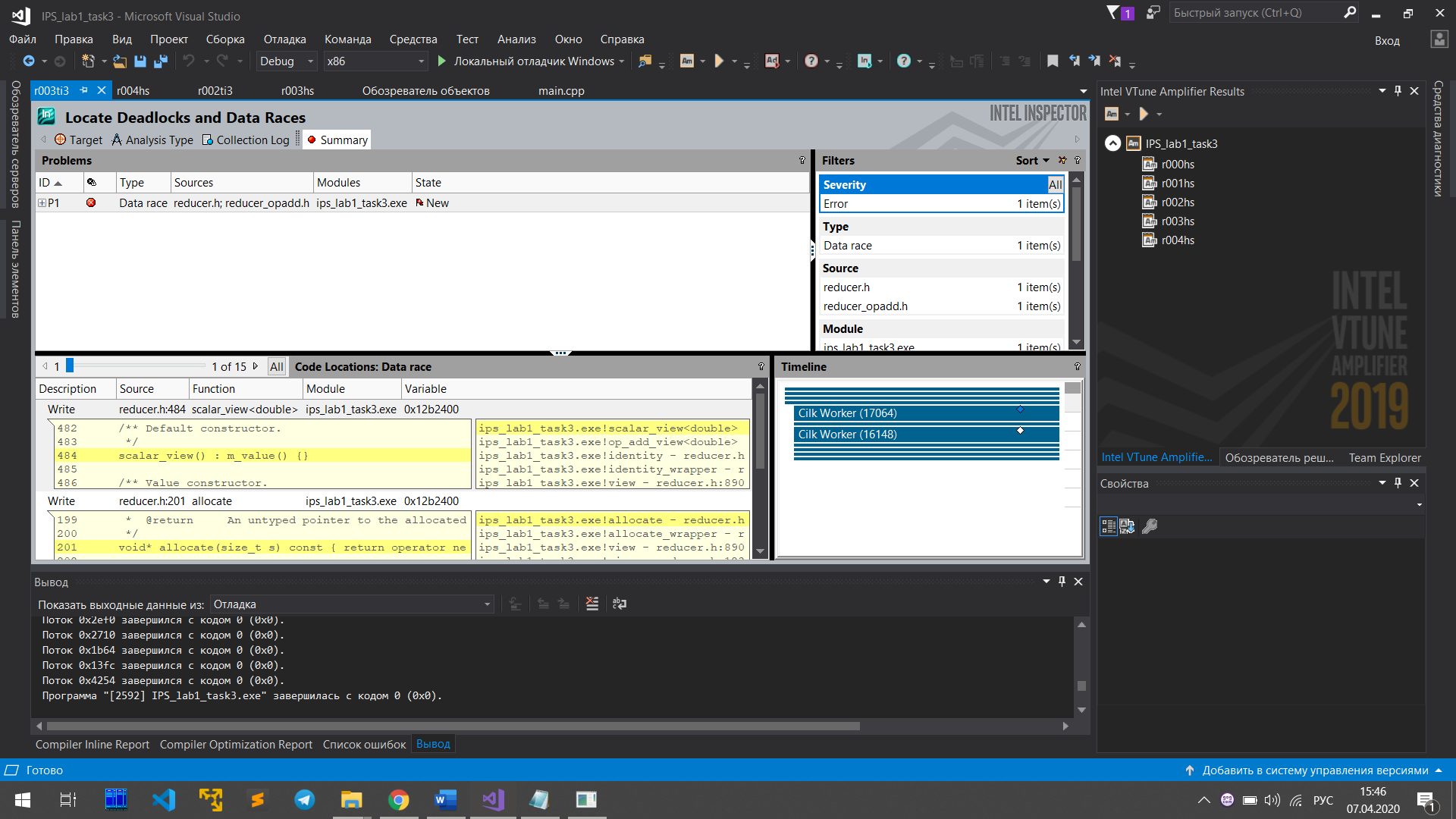


Рисунок 10 – пример работы Inspector на наличие гонок данных для программы с cilk\_for и добавленным reducer\_opadd

Как мы видим, добавление reducer\_opadd решило проблему с большим количеством гонок данных, reducer\_opadd указывает на одну гонку данных

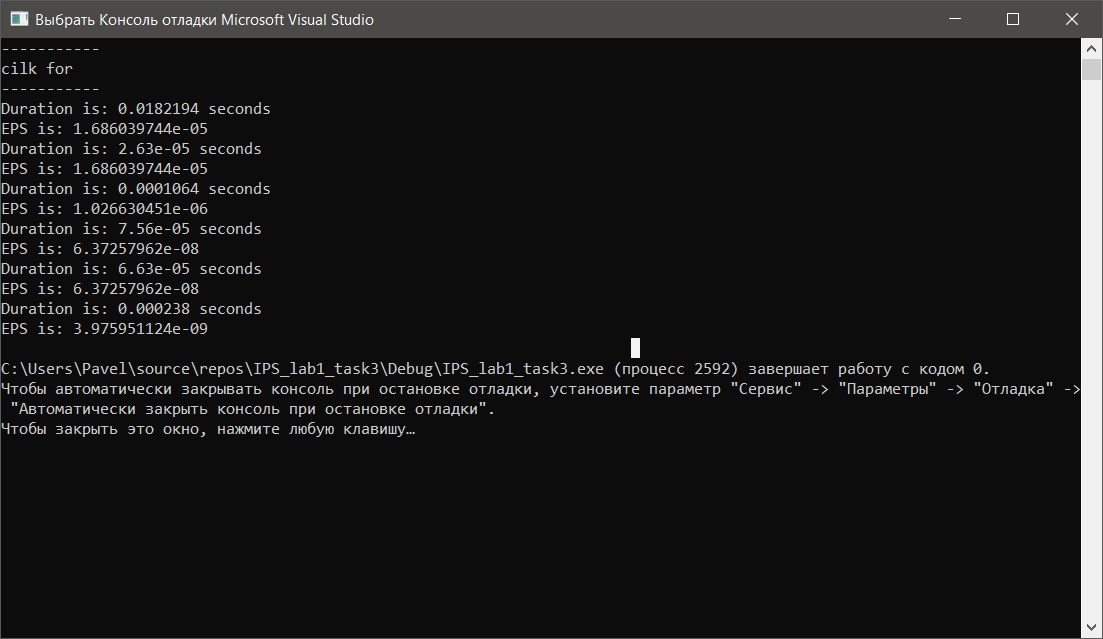


Рисунок 11 – результаты работ программы с cilk\_for и reducer\_opadd

Программа с использованием шаблонов TBB

Для использования шаблонов TBB пришлось переписать алгоритм вычисление интеграла по методу Симпсона, код приведен ниже

|  |
| --- |
| void simpsonIntegral(double eps) {  std::mutex gmutex;  double n = 1.0 / eps;  int a = -1, b = 1;  const double width = (b - a) / n;  double computedInt = 5.141592653589793;  cilk::reducer\_opadd<double> simpson\_integral(0);  high\_resolution\_clock::time\_point start = high\_resolution\_clock::now();  parallel\_for(blocked\_range<size\_t>(1, n), [&](const blocked\_range<size\_t>& r) {  for (int step = r.begin(); step < r.end(); step++) {  const double x1 = a + step \* width;  const double x2 = a + (step + 1)\*width;  gmutex.lock();  simpson\_integral += (x2 - x1) / 6.0\*(f(x1) + 4.0\*f(0.5\*(x1 + x2)) + f(x2));  gmutex.unlock();  }  });  high\_resolution\_clock::time\_point finish = high\_resolution\_clock::now();  duration<double> duration = (finish - start);  cout << "Duration is: " << duration.count() << " seconds" << endl;  cout << "EPS is: " << setprecision(10) << abs(computedInt - simpson\_integral.get\_value()) << endl;  } |

При использовании VTune Amplifier возникла следующая ошибка, переустановка программы не помогла, на других проекта программа работала, пофиксить проблему не удалось.

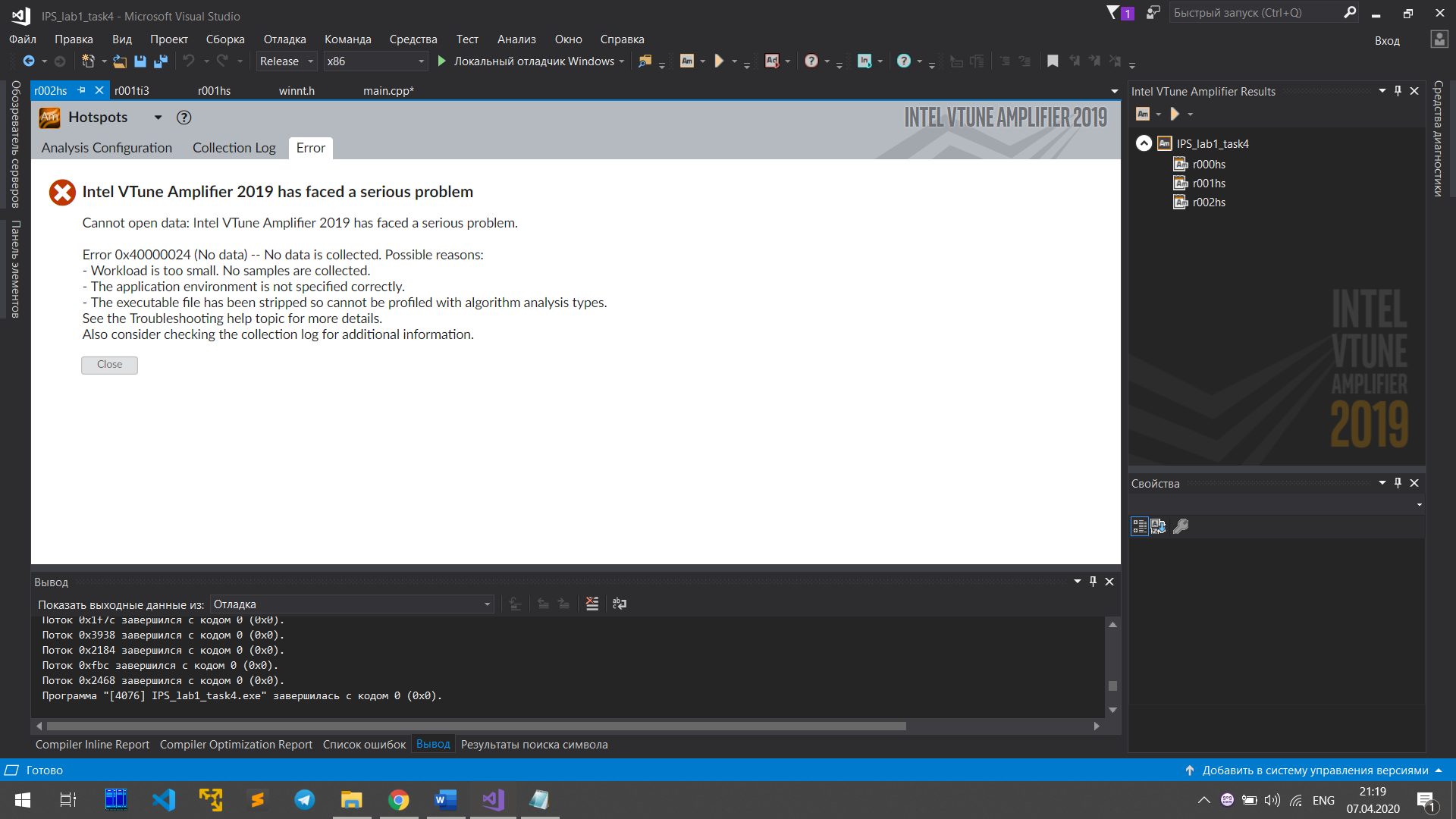


Рисунок 12 - пример работы VTune для программы с TBB

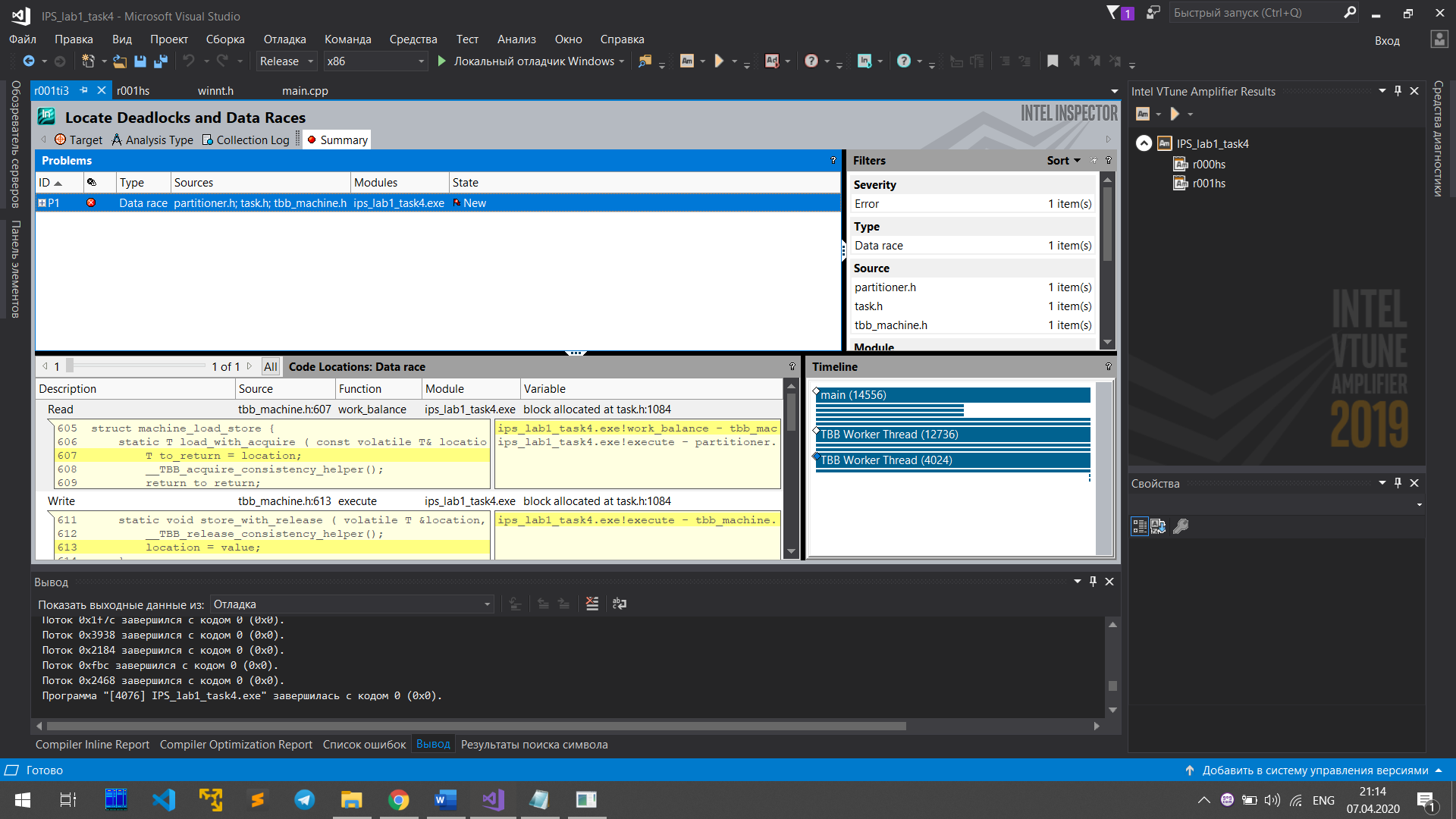


Рисунок 13 – пример работы Inspector на наличие гонок данных для программы с TBB

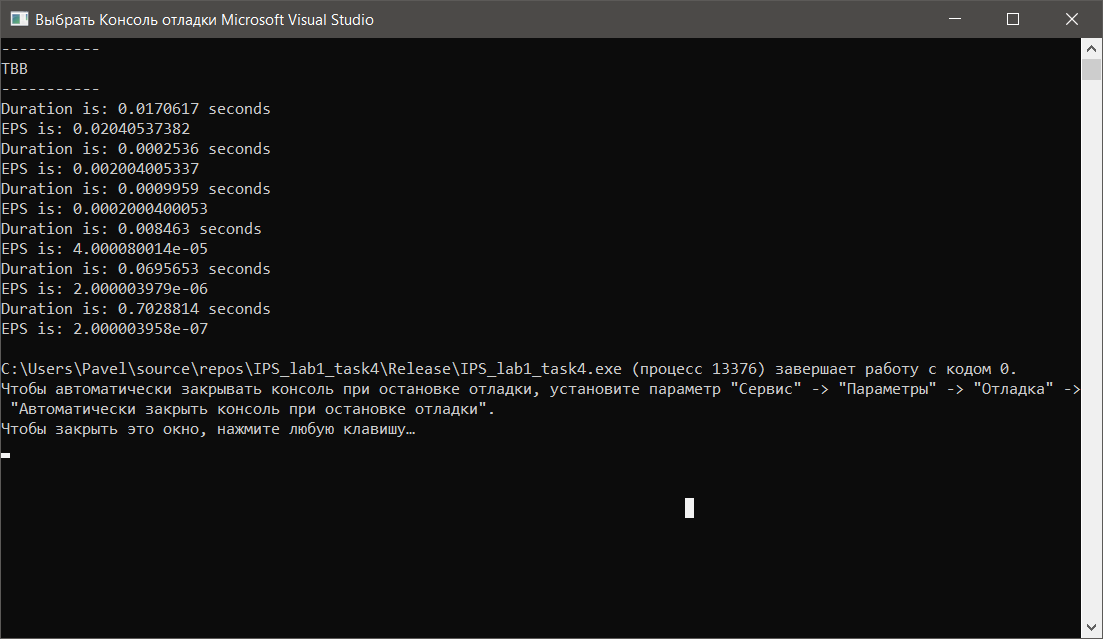


Рисунок 14 – результаты работ программы с cilk\_for и reducer\_opadd

Можно видеть, что результаты вычисления изменились не сильно, а затраченное время можно объяснить тем, что в исходном коде использовался mutex, чтобы обеспечить корректное вычисление интеграла, а операции связанные с mutex замедляют работу программы.