**Проектное задание**

**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: Ознакомьтесь со статьей**[**The non-uniform covering approach to manipulator workspace assessment.pdf**](https://canvas.instructure.com/courses/1844563/files/87817255/download?wrap=1)**[Предварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817255/download?wrap=1).**

**Задание 2: Скачайте следующие файлы:** [box.h](https://canvas.instructure.com/courses/1844563/files/87817219/download?wrap=1" \o "box.h)[Предварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817219/download?wrap=1), [box.cppПредварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817236/download?wrap=1), [fragmentation.h](https://canvas.instructure.com/courses/1844563/files/87817256/download?wrap=1" \o "fragmentation.h)[Предварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817256/download?wrap=1), [fragmentation.cppПредварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817228/download?wrap=1), [NUCovering.cppПредварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817257/download?wrap=1). **В этих файлах представлен предлагаемый каркас разрабатываемого проекта. Ознакомьтесь с содержимым каждого файла. После выполнения п.1. Вашей задачей является написание определений тех функций проекта, в теле которых представлен комментарий "// необходимо определить функцию".**

Добавленные участки кода:

1)void low\_level\_fragmentation::VerticalSplitter(const Box& box, boxes\_pair& vertical\_splitter\_pair):

void low\_level\_fragmentation::VerticalSplitter(const Box& box, boxes\_pair& vertical\_splitter\_pair)

{

double del = 2.0;

double x\_min, y\_min, width, height, newleft1, newtop1, newwidth1, newheight1, newleft2, newtop2, newwidth2, newheight2;

box.GetParameters(x\_min, y\_min, width, height);

newleft1 = x\_min;

newtop1 = y\_min;

newwidth1 = width / del;

newheight1 = height;

Box leftbox(newleft1, newtop1, newwidth1, newheight1);

newleft2 = x\_min + width / del;

newtop2 = y\_min;

newwidth2 = width / del;

newheight2 = height;

Box rightbox(newleft2, newtop2, newwidth2, newheight2);

vertical\_splitter\_pair.first = leftbox;

vertical\_splitter\_pair.second = rightbox;

}

2) void low\_level\_fragmentation::HorizontalSplitter(const Box& box, boxes\_pair& horizontal\_splitter\_pair):

double del = 2.0;

double x\_min, y\_min, width, height, newleft1, newtop1, newwidth1, newheight1, newleft2, newtop2, newwidth2, newheight2;

box.GetParameters(x\_min, y\_min, width, height);

newleft1 = x\_min;

newtop1 = y\_min;

newwidth1 = width;

newheight1 = height / del;

Box topbox(newleft1, newtop1, newwidth1, newheight1);

newleft2 = x\_min;

newtop2 = y\_min + height / del;

newwidth2 = width;

newheight2 = height / del;

Box bottombox(newleft2, newtop2, newwidth2, newheight2);

horizontal\_splitter\_pair.first = topbox;

horizontal\_splitter\_pair.second = bottombox;

3) void low\_level\_fragmentation::GetNewBoxes(const Box& box, boxes\_pair& new\_pair\_of\_boxes):

double width, height;

box.GetWidhtHeight(width, height);

if (abs(width) > abs(height))

VerticalSplitter(box, new\_pair\_of\_boxes);

else

HorizontalSplitter(box, new\_pair\_of\_boxes);

4) int low\_level\_fragmentation::ClasifyBox(const min\_max\_vectors& vects):

int count = 0;

for (int i = 0; i < vects.second.size(); i++)

{

if (vects.second[i] < 0)

count += 1;

if (vects.first[i] > 0)

return 1;

}

if (count == vects.second.size())

return 0;

if (vects.first[0] == 0 && vects.second[0] == 0)

return 2;

return 3;

5) void low\_level\_fragmentation::GetBoxType(const Box& box):

void low\_level\_fragmentation::GetBoxType(const Box& box)

{

min\_max\_vectors min\_max\_vecs;

boxes\_pair new\_pair\_of\_boxes;

GetMinMax(box, min\_max\_vecs);

int res = ClasifyBox(min\_max\_vecs);

switch (res)

{

case 0: {solution.push\_back(box); break; }

case 1: {not\_solution.push\_back(box); break; }

case 2: {boundary.push\_back(box); break; }

case 3:

{

GetNewBoxes(box, new\_pair\_of\_boxes);

temporary\_boxes.push\_back(new\_pair\_of\_boxes.first);

temporary\_boxes.push\_back(new\_pair\_of\_boxes.second);

break;

}

}

}

6) void high\_level\_analysis::GetSolution():

void high\_level\_analysis::GetSolution()

{

current\_box = Box(-g\_l1\_max, 0, g\_l2\_max + g\_l0 + g\_l1\_max, \_\_min(g\_l1\_max, g\_l2\_max));

std::vector<Box> current\_boxes;

temporary\_boxes.push\_back(current\_box);

int level = FindTreeDepth();

for (int i = 0; i < (level + 1); ++i)

{

//temporary\_boxes.move\_out(buf\_boxes);

buf\_boxes.assign(temporary\_boxes.begin(), temporary\_boxes.end());

current\_boxes = buf\_boxes;

buf\_boxes.clear();

for(int j = 0; j < current\_boxes.size(); ++j)

GetBoxType(current\_boxes[j]);

}

}

7) void WriteResults(const char\* file\_names[]):

void WriteResults(const char\* file\_names[])

{

double x\_min, y\_min, width, height;

vector <Box> temp;

std::ofstream fsolution(file\_names[0]);

std::ofstream fboundary(file\_names[1]);

std::ofstream fnot\_solution(file\_names[2]);

//solution.move\_out(temp);

temp.assign(solution.begin(),solution.end());

for (int i = 0; i < temp.size(); i++)

{

temp[i].GetParameters(x\_min, y\_min, width, height);

fsolution << x\_min << " " << y\_min << " " << width << " " << height << '\n';

}

temp.clear();

//boundary.move\_out(temp);

temp.assign(boundary.begin(), boundary.end());

for (int i = 0; i < temp.size(); i++)

{

temp[i].GetParameters(x\_min, y\_min, width, height);

fboundary << x\_min << " " << y\_min << " " << width << " " << height << '\n';

}

temp.clear();

//not\_solution.move\_out(temp);

temp.assign(not\_solution.begin(), not\_solution.end());

for (int i = 0; i < temp.size(); i++)

{

temp[i].GetParameters(x\_min, y\_min, width, height);

fnot\_solution << x\_min << " " << y\_min << " " << width << " " << height << '\n';

}

fsolution.close();

fboundary.close();

fnot\_solution.close();

}

А также файл NUCovering.cpp, содержащий функцию main:

#include "fragmentation.h"

#include <locale.h>

/// параметры начальной прямоугольной области

/\*

const double g\_l1\_max = 12.0;

const double g\_l2\_max = g\_l1\_max;

const double g\_l1\_min = 8.0;

const double g\_l2\_min = g\_l1\_min;

const double g\_l0 = 5.0;

/// точность аппроксимации рабочего пространства

const double g\_precision = 0.25;

\*/

int main()

{

setlocale(LC\_ALL, "Rus");

high\_level\_analysis main\_object;

high\_resolution\_clock::time\_point start = high\_resolution\_clock::now();

main\_object.GetSolution();

high\_resolution\_clock::time\_point finish = high\_resolution\_clock::now();

duration<double> duration = (finish - start);

cout << "Precision: " << g\_precision << endl;

cout << "Duration: " << duration.count() << " seconds" << endl;

const char\* out\_files[3] = { "Solution.txt", "Boundary.txt", "Not\_Solution.txt" };

WriteResults(out\_files);

return 0;

}

**Задание 3: Реализация последовательной версии программы, определяющей рабочее пространство планарного робота, по предложенному в статье из п.1. алгоритму. Функция WriteResults() должна записывать значения параметров box-ов в выходные файлы в следующем порядке: x\_min, y\_min, width, *height*, *'\n'*. На выходе из программы должно получиться  3 файла. Определите время работы последовательной версии разработанной программы в двух режимах: Debug и Release. Сделайте скрины консоли, где отображается время работы для обоих случаев. Вставьте скрины в отчет к проекту, дав им соответствующие названия. Постройте полученное рабочее пространство, используя скрипт MATLAB** [PrintWorkspace.m](https://canvas.instructure.com/courses/1844563/files/87817241/download?wrap=1" \o "PrintWorkspace.m)[Предварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817241/download?wrap=1). **Сохраните изображение рабочего пространства. Вставьте его в отчет, назвав соответствующим образом.**

Скорости выполнения в режиме:

1) Debug x64:

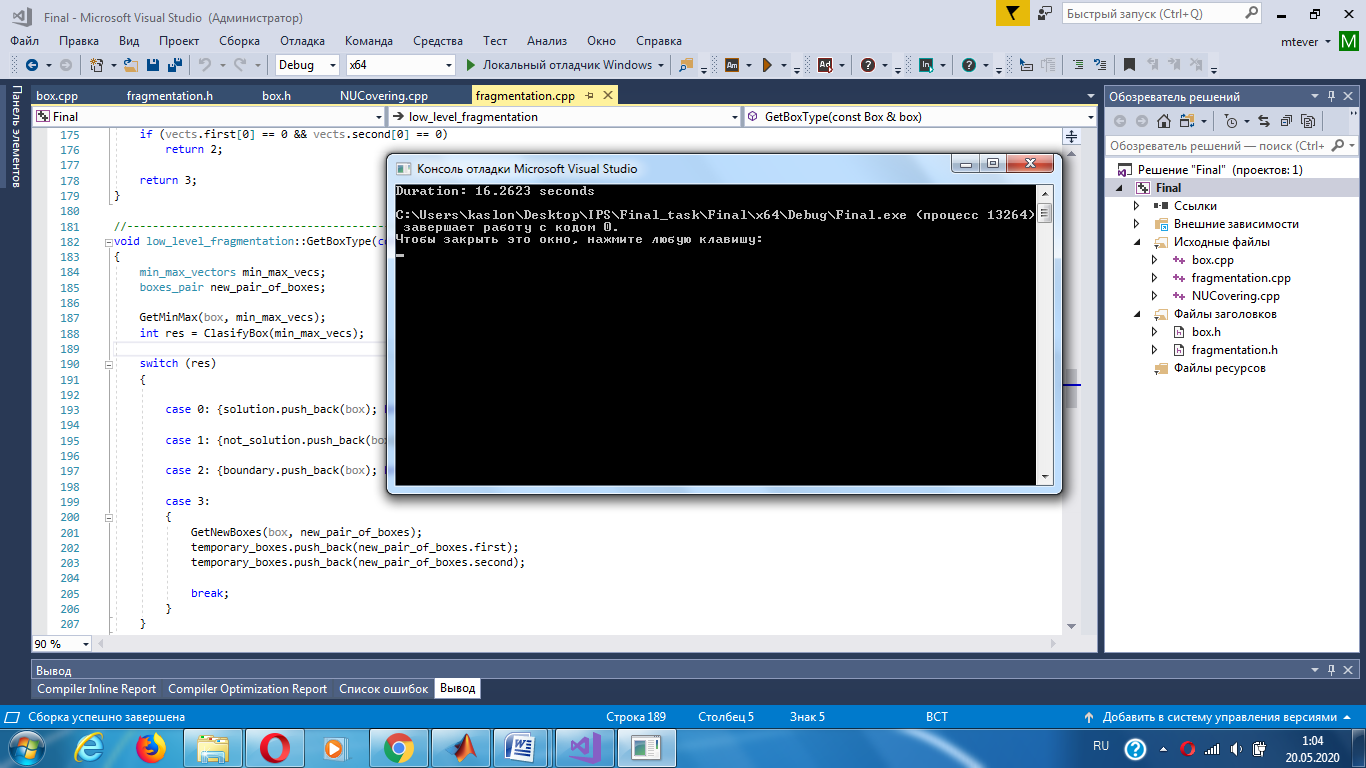


Рисунок 1. – Скриншот результата в режиме Debug

2) Release x64:

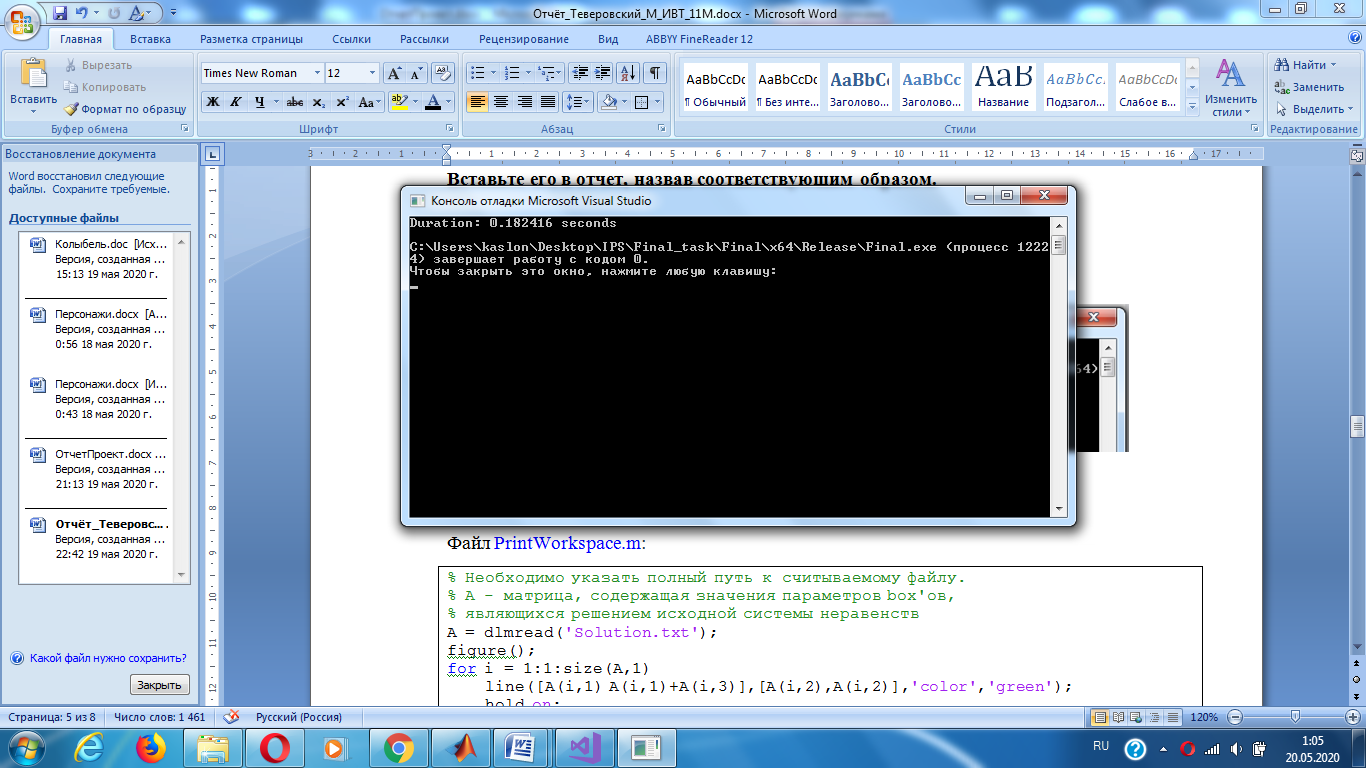


Рисунок 2. – Скриншот результата в режиме Release

**Анализ полученных результатов**:

Получили более 16 секунд в режиме Debug против 0.18 секунд в режиме Release.

Возможно это связано с тем, что оптимизация ещё не отключалась.

Файл[PrintWorkspace.m](https://canvas.instructure.com/courses/1844563/files/87817241/download?wrap=1):

% Необходимо указать полный путь к считываемому файлу.

% А - матрица, содержащая значения параметров box'ов,

% являющихся решением исходной системы неравенств

A = dlmread('Solution.txt');

figure();

for i = 1:1:size(A,1)

line([A(i,1) A(i,1)+A(i,3)],[A(i,2),A(i,2)],'color','green');

hold on;

line([A(i,1) A(i,1)],[A(i,2),A(i,2)+A(i,4)],'color','green');

line([A(i,1) A(i,1)+A(i,3)],[A(i,2)+A(i,4),A(i,2)+A(i,4)],'color','green');

line([A(i,1)+ A(i,3),A(i,1)+ A(i,3)],[A(i,2),A(i,2)+A(i,4)],'color','green');

end

axis([-12 17 0 12]);

% Необходимо указать полный путь к считываемому файлу.

% B - матрица, содержащая значения параметров box'ов,

% находящихся на границе между множеством решений исходной системы

% и множеством, не являющимся решением

B = dlmread('Boundary.txt');

for i = 1:1:size(B,1)

line([B(i,1) B(i,1)+B(i,3)],[B(i,2),B(i,2)],'color','red');

line([B(i,1) B(i,1)],[B(i,2),B(i,2)+B(i,4)],'color','red');

line([B(i,1) B(i,1)+B(i,3)],[B(i,2)+B(i,4),B(i,2)+B(i,4)],'color','red');

line([B(i,1)+ B(i,3),B(i,1)+ B(i,3)],[B(i,2),B(i,2)+B(i,4)],'color','red');

end

% Необходимо указать полный путь к считываемому файлу.

% С - матрица, содержащая значения параметров box'ов,

% не являющихся решением исходной системы неравенств

C = dlmread('Not\_Solution.txt');

for i = 1:1:size(C,1)

line([C(i,1) C(i,1)+C(i,3)],[C(i,2),C(i,2)]);

line([C(i,1) C(i,1)],[C(i,2),C(i,2)+C(i,4)]);

line([C(i,1) C(i,1)+C(i,3)],[C(i,2)+C(i,4),C(i,2)+C(i,4)]);

line([C(i,1)+ C(i,3),C(i,1)+ C(i,3)],[C(i,2),C(i,2)+C(i,4)]);

end

title('Manipulator workspace, \delta = 0.25');

xlabel('x'); ylabel('y');



Рисунок 3. – Скриншот полученного рабочего пространства по заданию 3

**Задание 4: Использование *Amplifier XE* в целях определения наиболее часто используемых участков кода. Для этого закомментируйте строки кода, отвечающие за запись результатов в выходные файлы, выберите *New Analysis* из меню *Amplifier XE* на панели инструментов, укажите тип анализа *Basic Hotspots*, запустите анализ. Сделайте скрин окна результатов анализа и вкладки Bottom-up*.*В списке, представленном в разделе *Top Hotspots*вкладки Summary должна фигурировать функция *GetMinMax()*.**

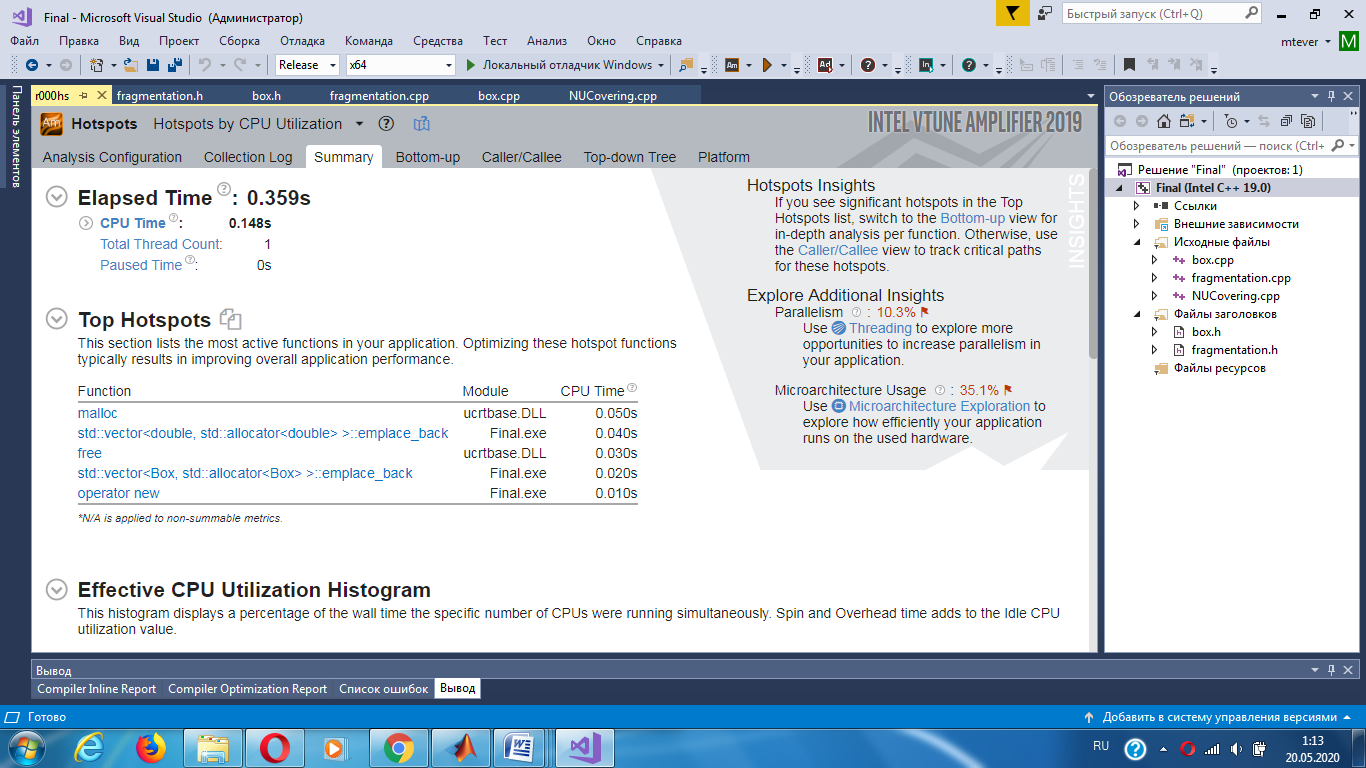
****

Рисунок 4. – Скриншот результатов Amplifier XE. Summary

В окне Bottom-up как раз фигурирует функция GetMinMax():

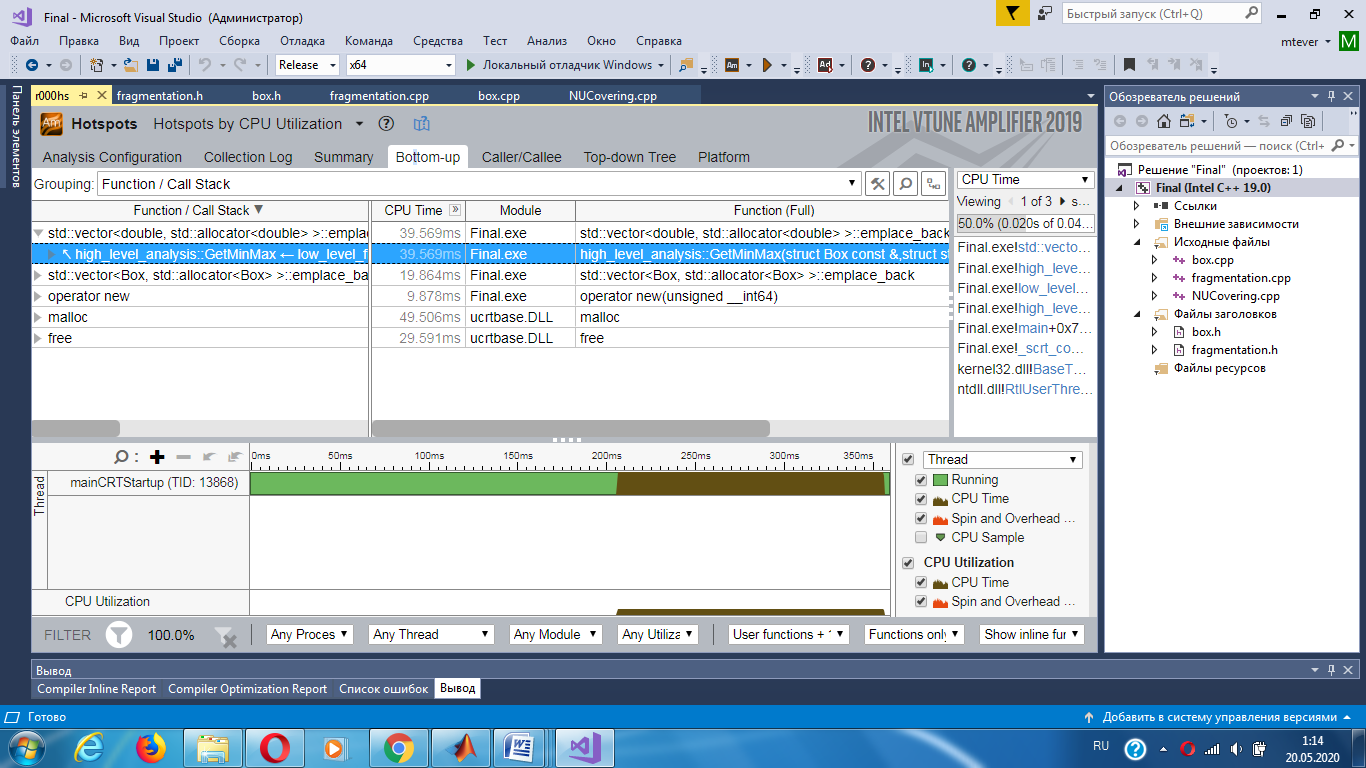
****

Рисунок 5. – Скриншот результатов Amplifier XE. Bottom-up

**Задание 5: Использование *Parallel Advisor* с целью определения участков кода, которые требуют наибольшего времени исполнения. Переведите проект в режим Release и отключите всякую оптимизацию. Для этого следует выбрать свойства проекта, во вкладке *С\С++* перейти в раздел *Оптимизация*, в пункте меню *“Оптимизация”* выбрать *Отключено (/Od)*. Далее выберем *Parallel Advisor* на панели инструментов Visual Studio и запустим *Survey Analysis*. По окончанию анализа Вы должны увидеть, что наибольшее время затрачивается в цикле функции *GetSolution()*, двойным кликом по данной строке отчета можно перейти к участку исходного кода и увидеть, что имеется в виду цикл, в котором на каждой итерации вызывается функция *GetBoxType()*. Сделайте скрины результатов *Survey Analysis*, сохраните их, добавьте в отчет. Вернитесь в режим Debug.**

**1) Отключение оптимизации:**

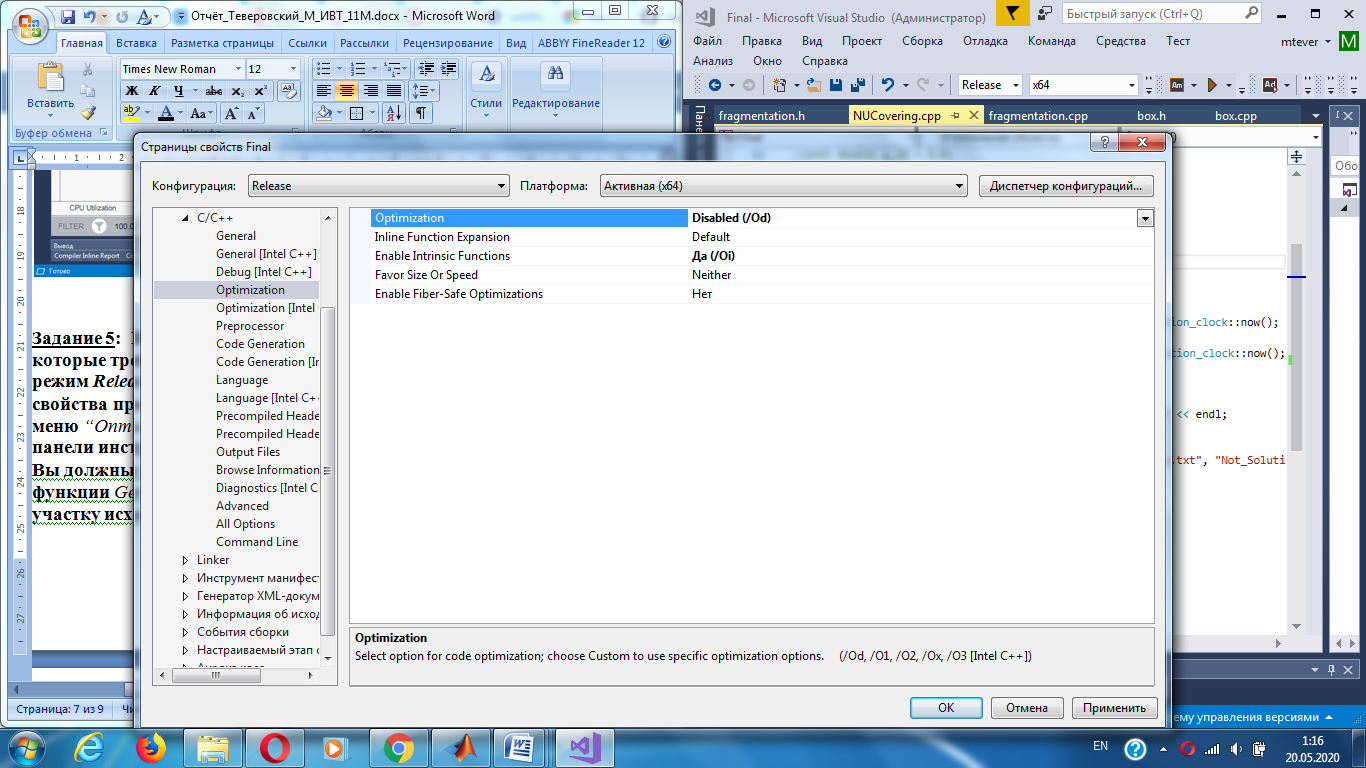


Рисунок 6. – Настройки проекта

**2) Parallel Advisor:**

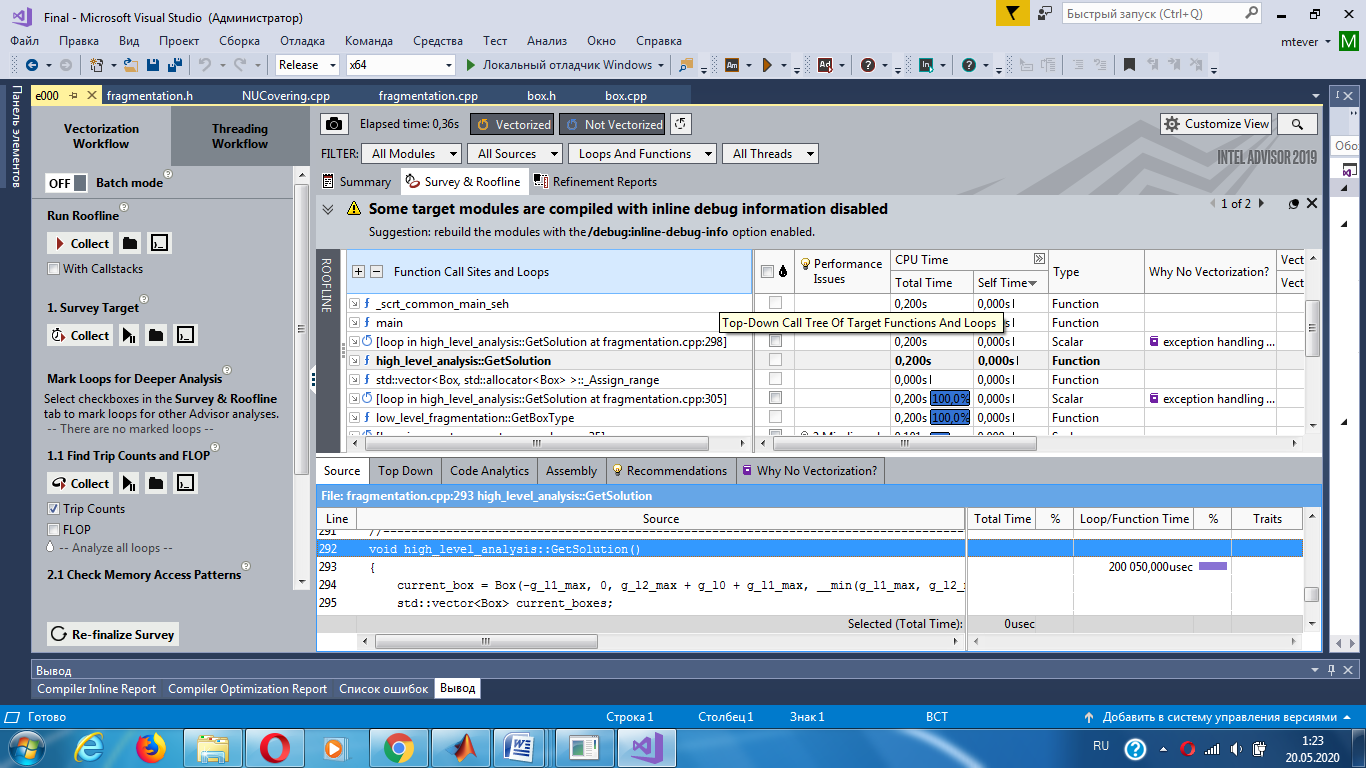


Рисунок 7. – Скриншот результатов Parallel Advisor

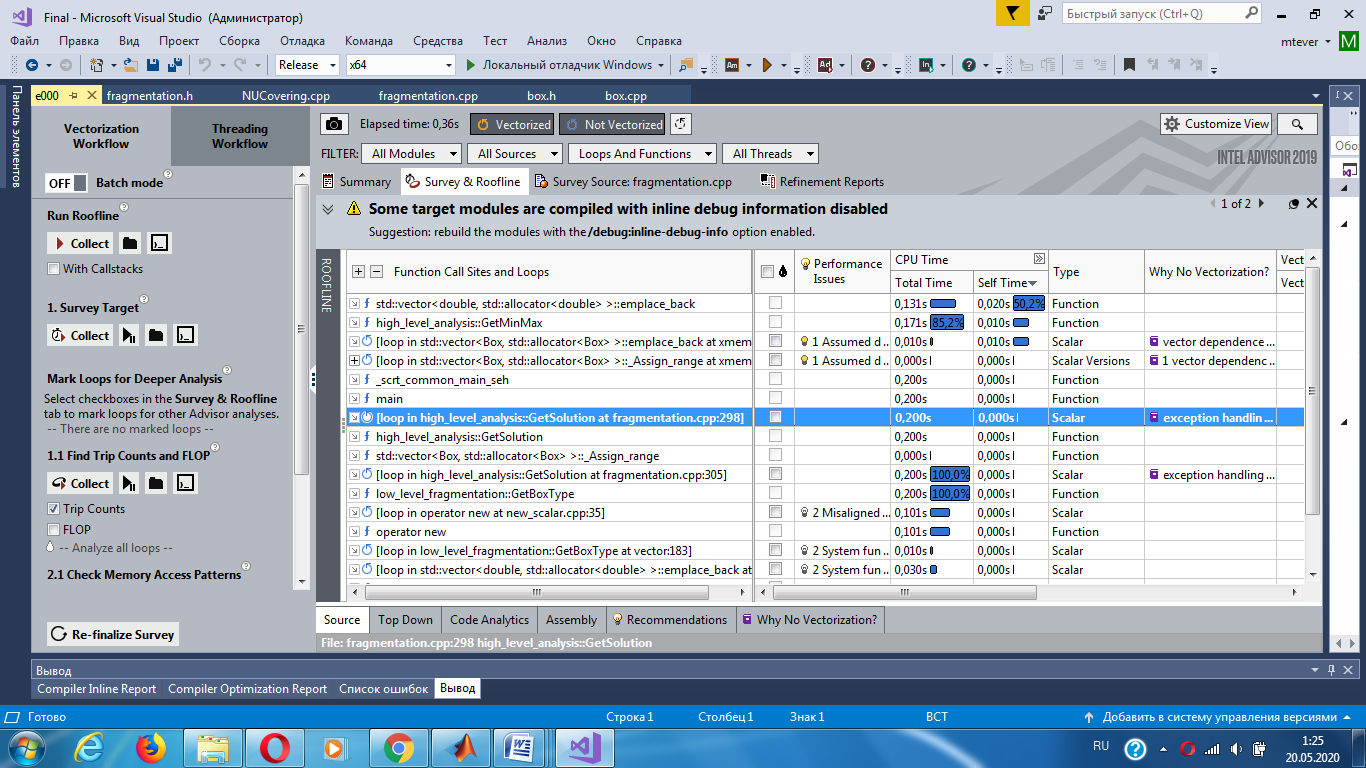


Рисунок 8. – Скриншот результатов Parallel Advisor

**Анализ полученных результатов**:

Действительно, наибольшее время затрачивается на цикл GetSolution(). Необходимо введение параллелизма.

**Задание 6: Введение параллелизма в программу. В текущей (последовательной) реализации программы, в функции *GetSolution()* должны фигурировать два вложенных цикла. Внешний цикл проходит по всем уровням двоичного дерева разбиения. В рамках внутреннего цикла происходит перебор всех box-ов текущего уровня разбиения и определение типа box-а (является он частью рабочего пространства либо не является, лежит он на границе или подлежит дальнейшему анализу). Вам необходимо ввести параллелизм во внутренний цикл. Тогда следует подумать о возможности независимого обращения к векторам solution, not\_solution, boundary, *temporary\_boxes*. Для этого предлагается использовать *reducer* векторы *Intel Cilk Plus*, вместо обычных *std::vector*’ов.**

Изменённый участок кода в файле fragmentation.cpp:

1) reduced:

std::vector<Box> buf\_boxes;

cilk::reducer< cilk::op\_vector<Box> > solution;

cilk::reducer< cilk::op\_vector<Box> > not\_solution;

cilk::reducer< cilk::op\_vector<Box> > boundary;

cilk::reducer< cilk::op\_vector<Box> > temporary\_boxes;

2) GetSolution():

void high\_level\_analysis::GetSolution()

{

current\_box = Box(-g\_l1\_max, 0, g\_l2\_max + g\_l0 + g\_l1\_max, \_\_min(g\_l1\_max, g\_l2\_max));

std::vector<Box> current\_boxes;

temporary\_boxes->push\_back(current\_box);

int level = FindTreeDepth();

for (int i = 0; i < (level + 1); ++i)

{

temporary\_boxes.move\_out(buf\_boxes);

//buf\_boxes.assign(temporary\_boxes.begin(), temporary\_boxes.end());

current\_boxes = buf\_boxes;

buf\_boxes.clear();

cilk\_for(int j = 0; j < current\_boxes.size(); ++j)

GetBoxType(current\_boxes[j]);

}

}

**Время выполнения программы в результате:**

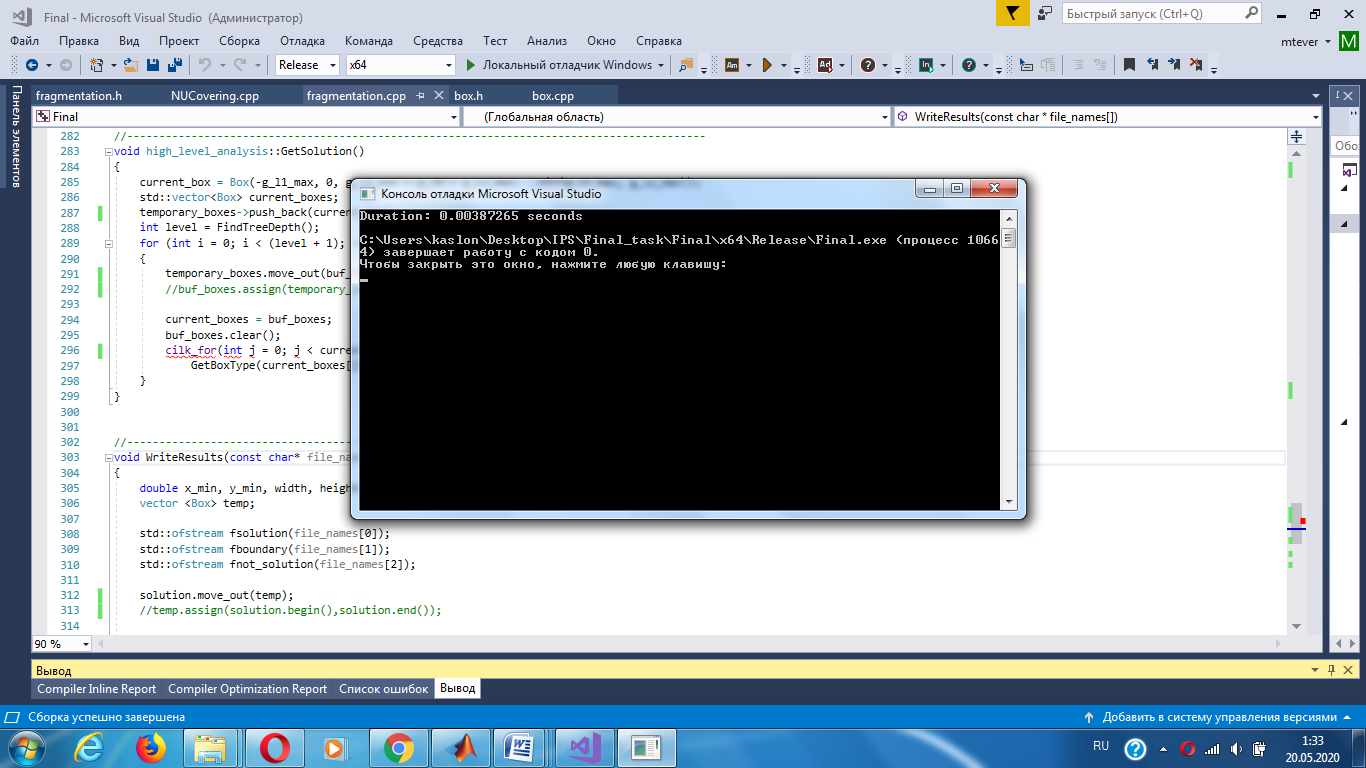


Рисунок 8. – Время выполнения кода с параллелизмом

\*Время выполнения сократилось до 3 милисекунд, тем не менее необходимы проверки на наличие гонок данных и иных ошибок при введении параллелизма в код.

**Задание 7: Определение ошибок после введения параллелизации. Запустите анализы  Inspector XE: Memory Error Analysis и Threading Error Analysis на различных уровнях (Narrowest, Medium, Widest). Приложите к отчету скрины результатов запуска перечисленных анализов. Исправьте обнаруженные ошибки, приложите новые скрины результатов анализов, в которых ошибки отсутствуют. Примечание: "глюки" Intel Cilk Plus исправлять не нужно.**

**1) Threading Error Analysis**

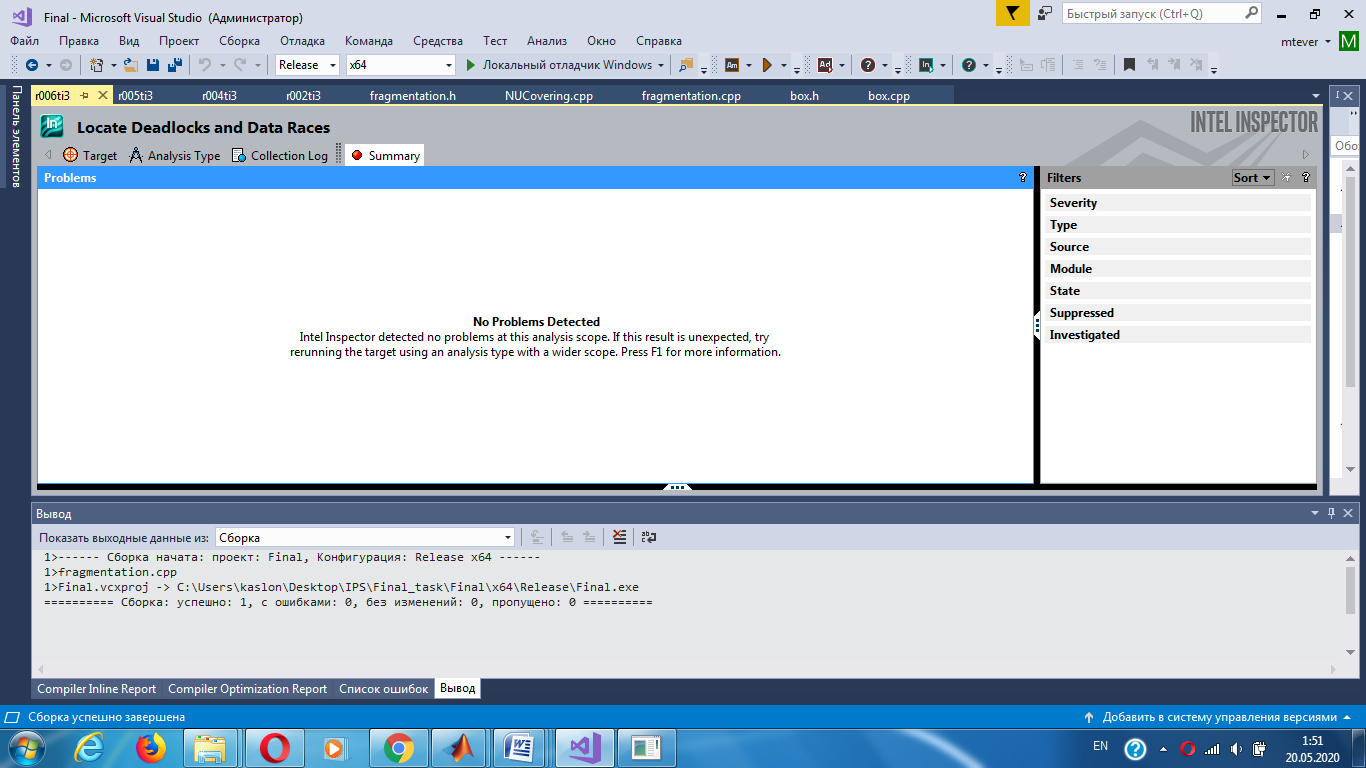
****

Рисунок 9 – Результаты работы Inspector XE.ThreadingErrorAnalysis

**Анализ полученных результатов**:

Гонок данных обнаружено не было, значит reducer используется верно

**2) Memory Error Analysis**

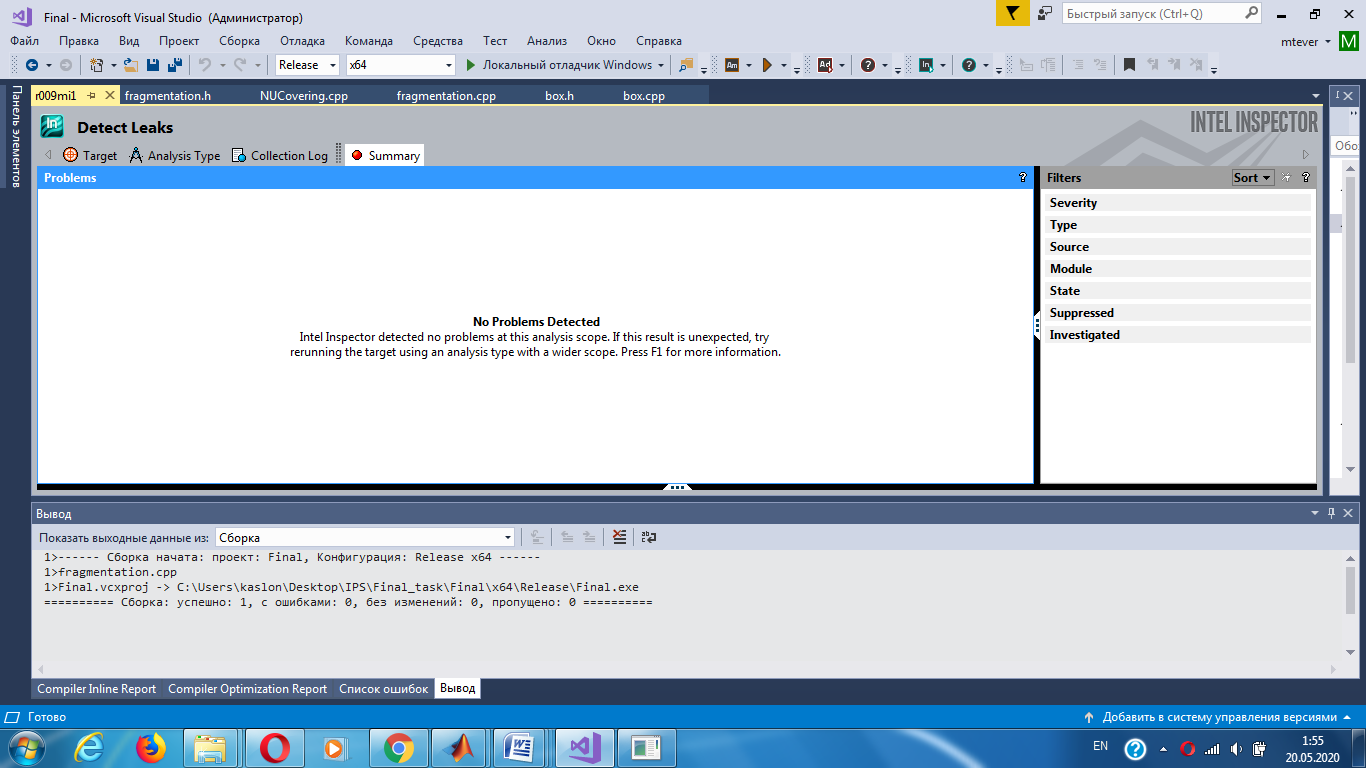
****

Рисунок 10 – Результаты работы Inspector XE. Detected Leaks

**3) Locate Memory Problems**

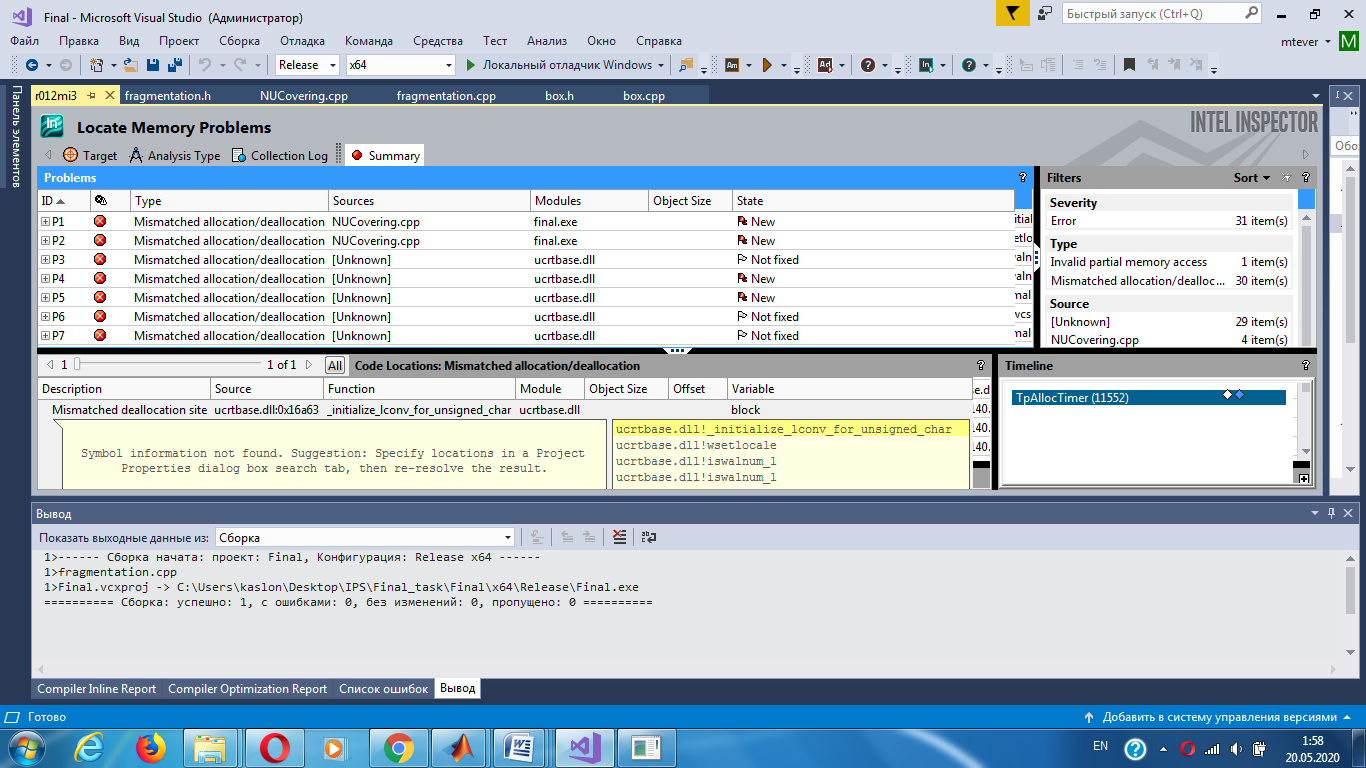
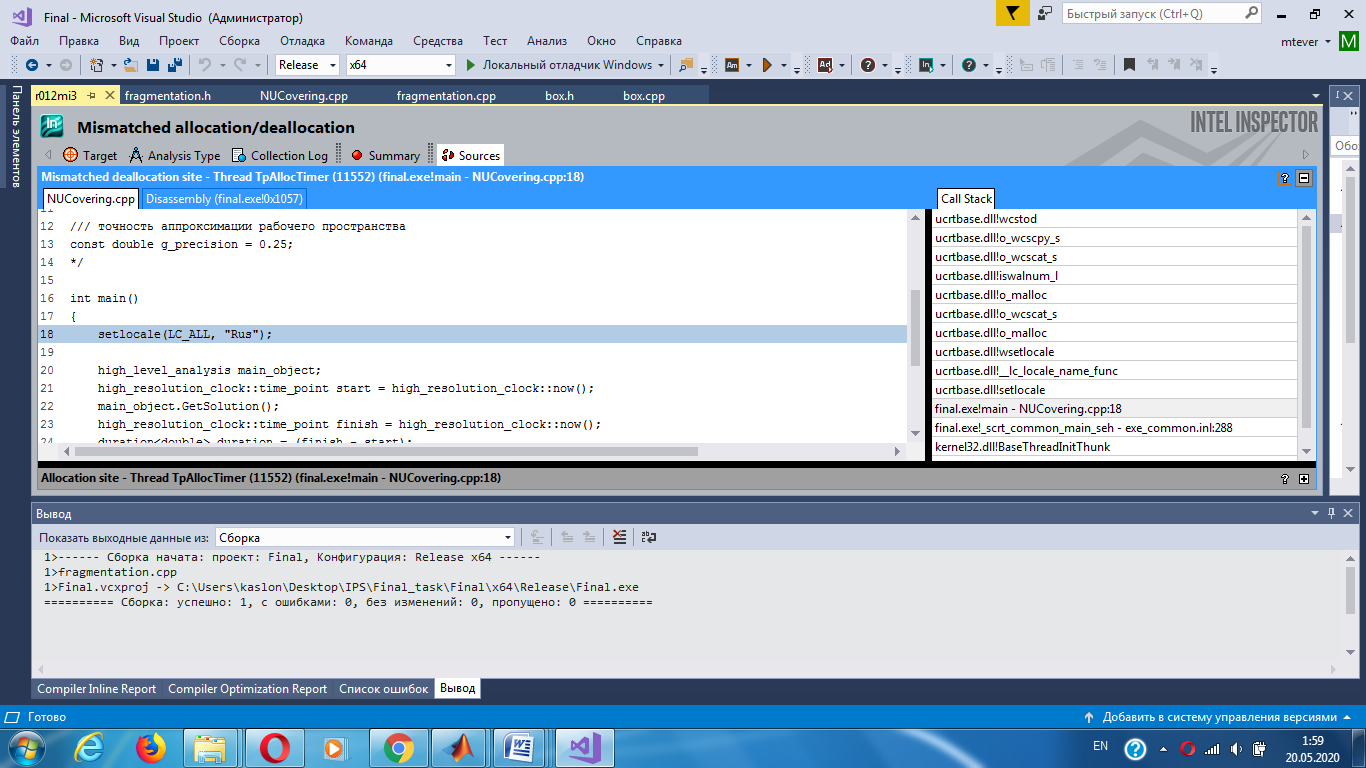


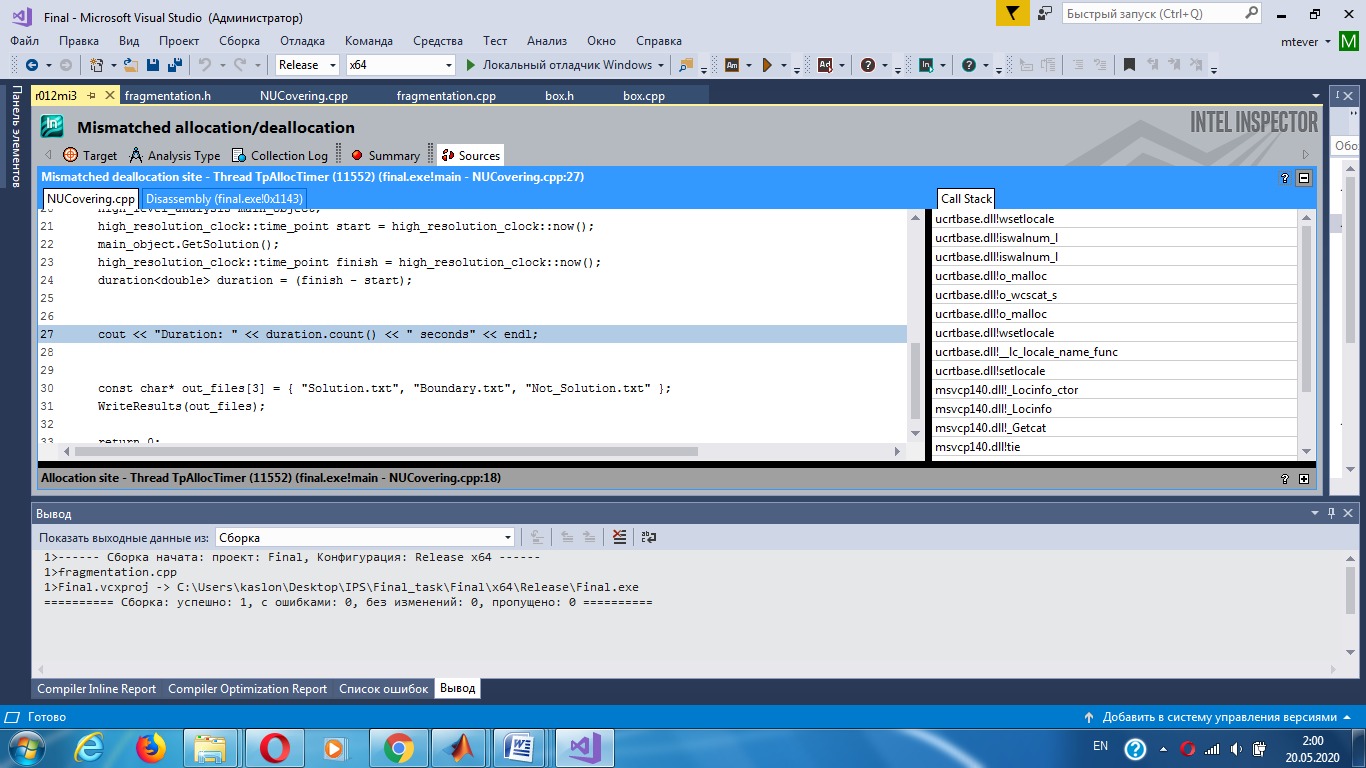
Рисунок 11 – Результаты работы Inspector XE. Locate Memory Problems

**Анализ полученных результатов**:

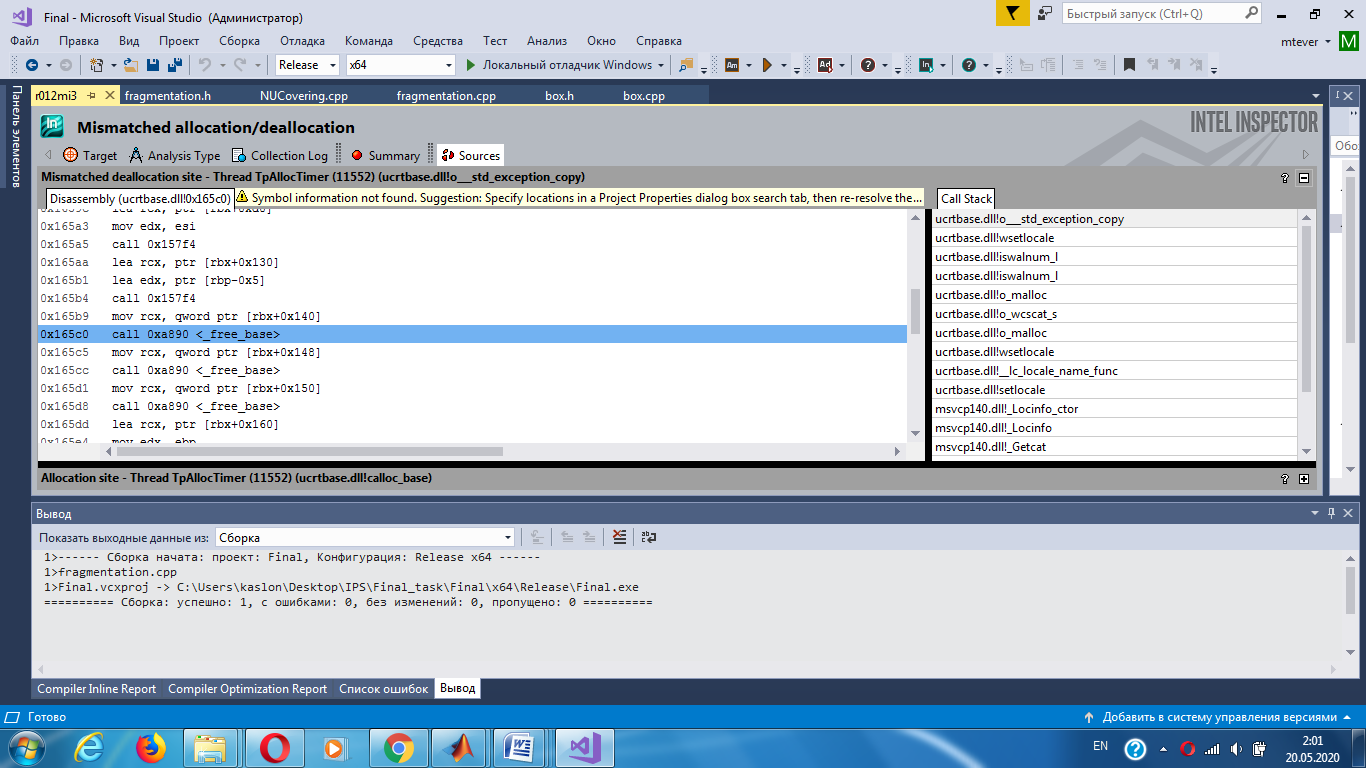
В первых двух ошибках ругается на setlocate:



И вывод времени выполнения программы:



Все остальные – на вызовы процедур.



Тем не менее, критических ошибок нет

**Задание 8: Работа с Cilk API. По умолчанию параллельная программа, использующая Cilk запускается на количестве потоков равных количеству ядер вашего компьютера. Для управления количеством вычислителей необходимо добавить заголовочных файл #include <cilk/cilk\_api.h> и действовать следующим образом: в исполняемом файле NUCovering.cpp перед созданием объекта main\_object  класса high\_level\_analysis необходимо вставить следующие строки кода: *\_\_cilkrts\_end\_cilk(); \_\_cilkrts\_set\_param("nworkers", "X");*Здесь Х - отвечает за количество вычислителей, на которых будет запускаться исходная программа. Это число может быть от 1 до N, где N - количество ядер в Вашей системе. Изменяя Х, запускайте программу и фиксируйте время ее выполнения, каждый раз сохраняйте скрины консоли*,* где должно быть отображено количество вычислителей (*cout << "Number of workers " << \_\_cilkrts\_get\_nworkers() << endl;*) и время работы программы.**

Изменённый файл NUCovering.cpp :

#include "fragmentation.h"

#include <locale.h>

#include <cilk/cilk\_api.h>

/// параметры начальной прямоугольной области

/\*

const double g\_l1\_max = 12.0;

const double g\_l2\_max = g\_l1\_max;

const double g\_l1\_min = 8.0;

const double g\_l2\_min = g\_l1\_min;

const double g\_l0 = 5.0;

/// точность аппроксимации рабочего пространства

const double g\_precision = 0.25;

\*/

int main()

{

setlocale(LC\_ALL, "Rus");

\_\_cilkrts\_end\_cilk(); \_\_cilkrts\_set\_param("nworkers", "2");

high\_level\_analysis main\_object;

high\_resolution\_clock::time\_point start = high\_resolution\_clock::now();

main\_object.GetSolution();

high\_resolution\_clock::time\_point finish = high\_resolution\_clock::now();

duration<double> duration = (finish - start);

cout << "Number of workers " << \_\_cilkrts\_get\_nworkers() << endl;

cout << "Duration: " << duration.count() << " seconds" << endl;

const char\* out\_files[3] = { "Solution.txt", "Boundary.txt", "Not\_Solution.txt" };

WriteResults(out\_files);

return 0;

}

К сожалению, в моём ноутбуке всего лишь 2 ядра, поэтому ниже представлены 2 попытки запуска:

1) При 2 ядрах:

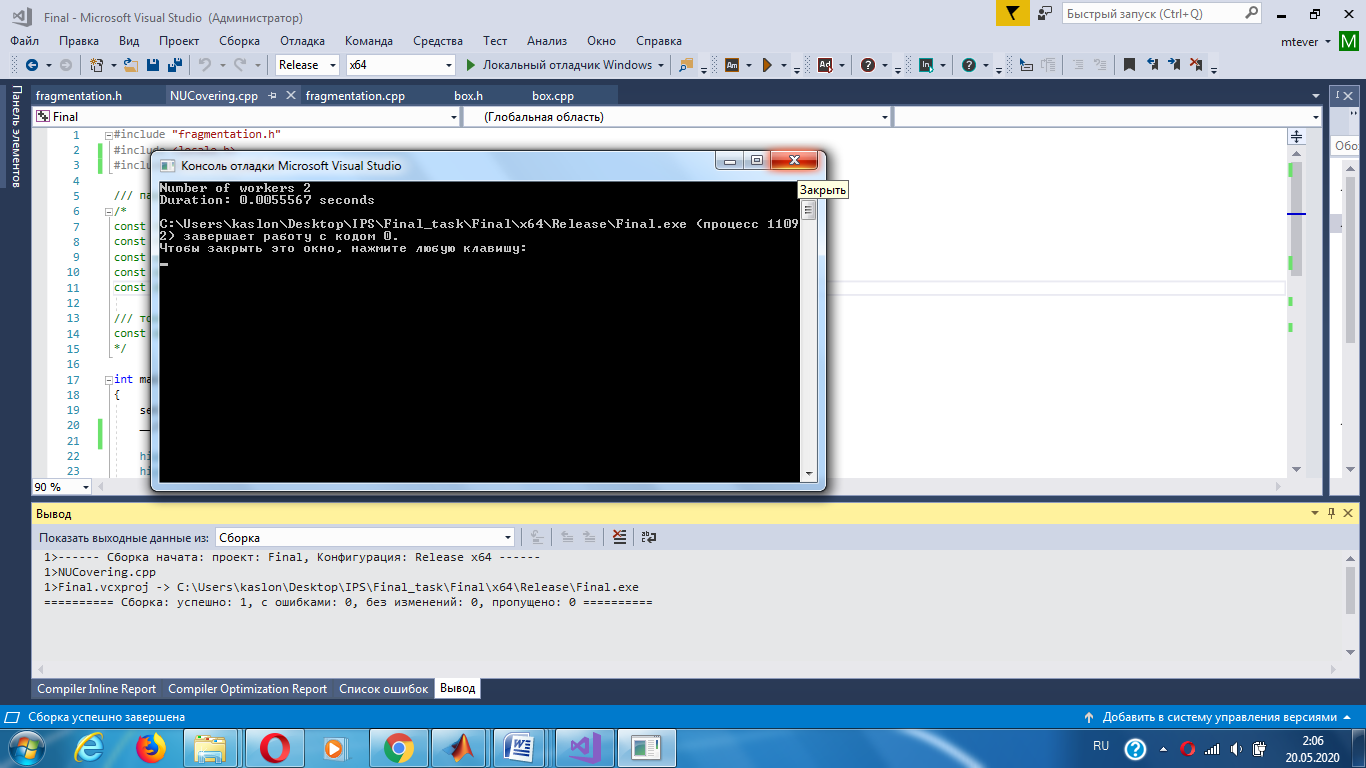


Рисунок 12 – Скорость выполнения при использовании 2 ядер

2) При 1 ядре:

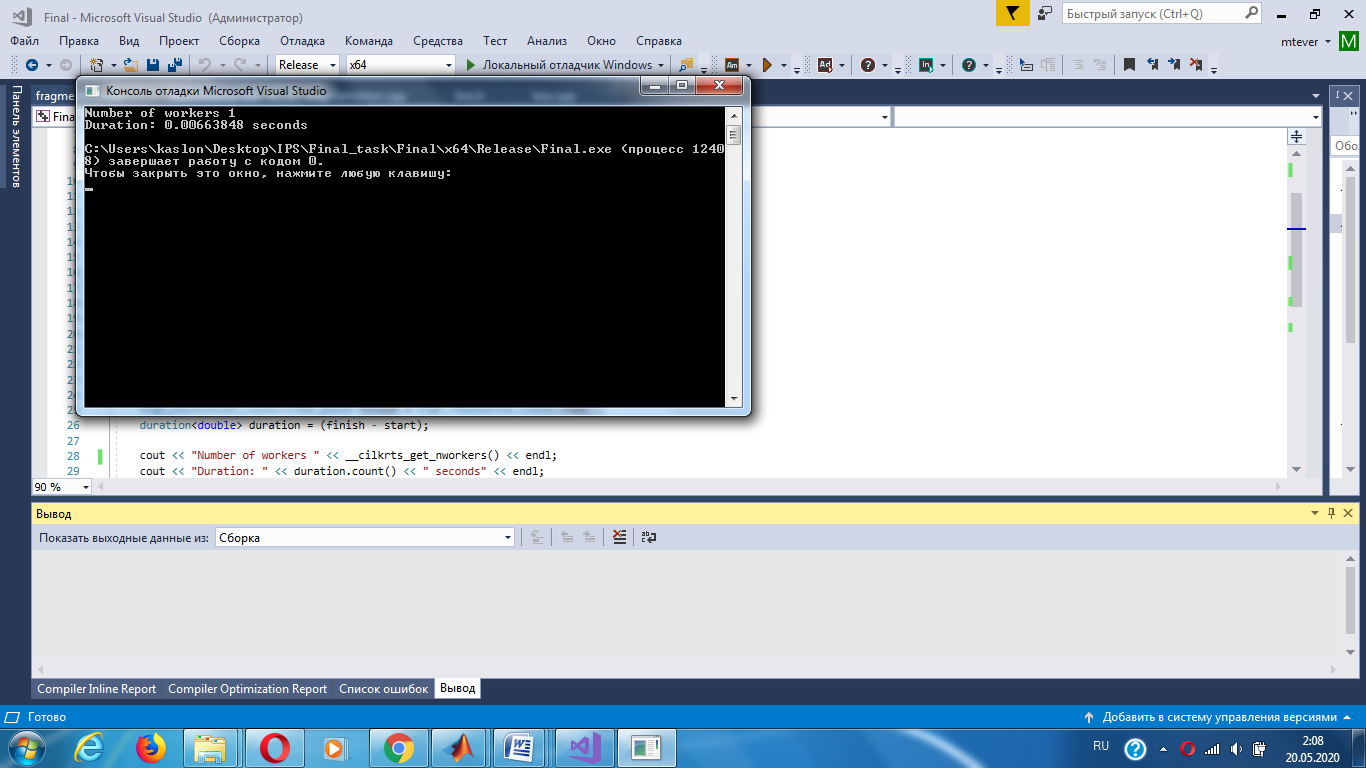


Рисунок 12 – Скорость выполнения при использовании 1 ядра

**Анализ полученных результатов**:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Количество ядер | 2 | 1 |
| Скорось выполнения | 0.0055567 | 0.00663848 |

Таблица 1 – Сравнение скорости выполнения

Примерно на 1 милисекунду выполнение при использовании 2 ядер быстрее

**Задание 10: Визуализация полученного решения. Поэкспериментируйте со входными параметрами программы и отобразите несколько версий полученного рабочего пространство робота. Рисунки приложите к отчету.**

Для сравнения везде будем использовать 2 ядра.

1) При delta = 0.25:

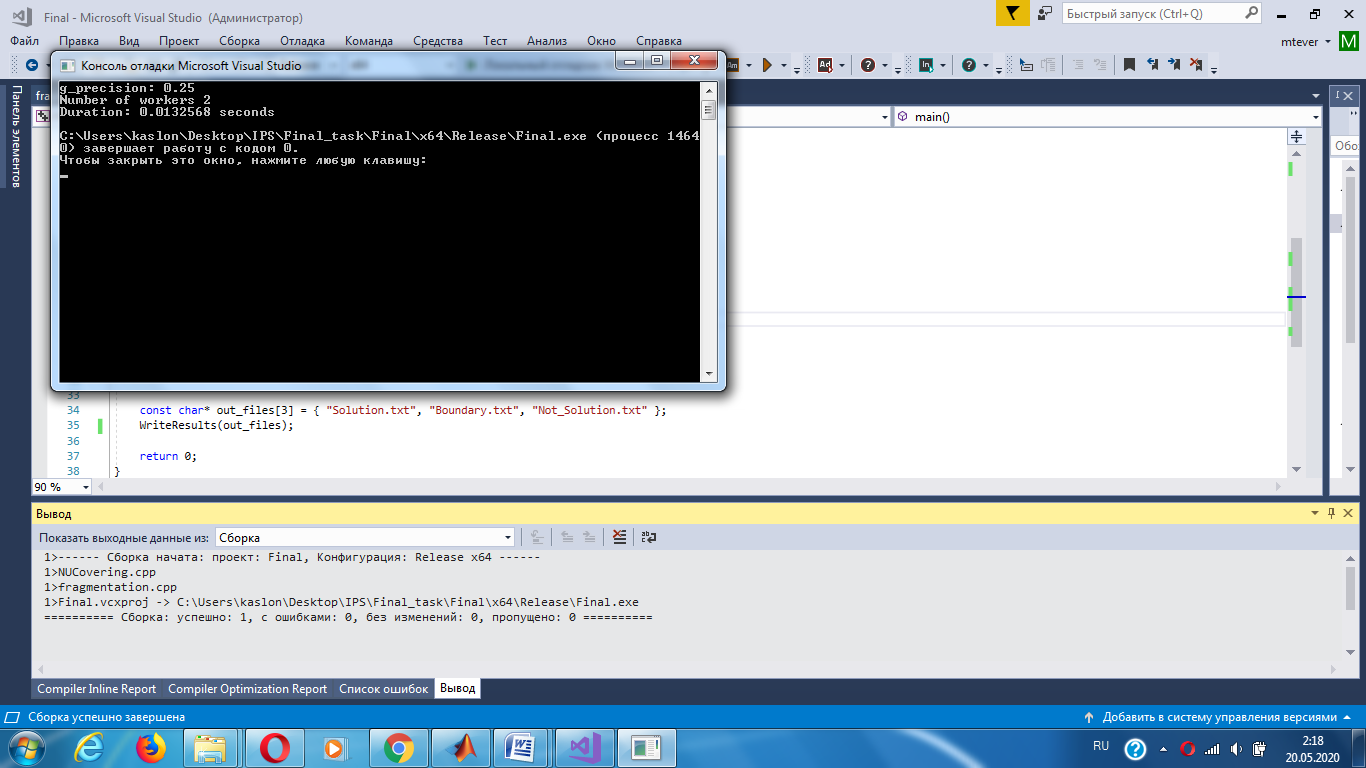
****

Рисунок 13 – Результат при delta = 0.25

****

Рисунок 14 – Результат при delta = 0.25

2) При delta = 0.5:

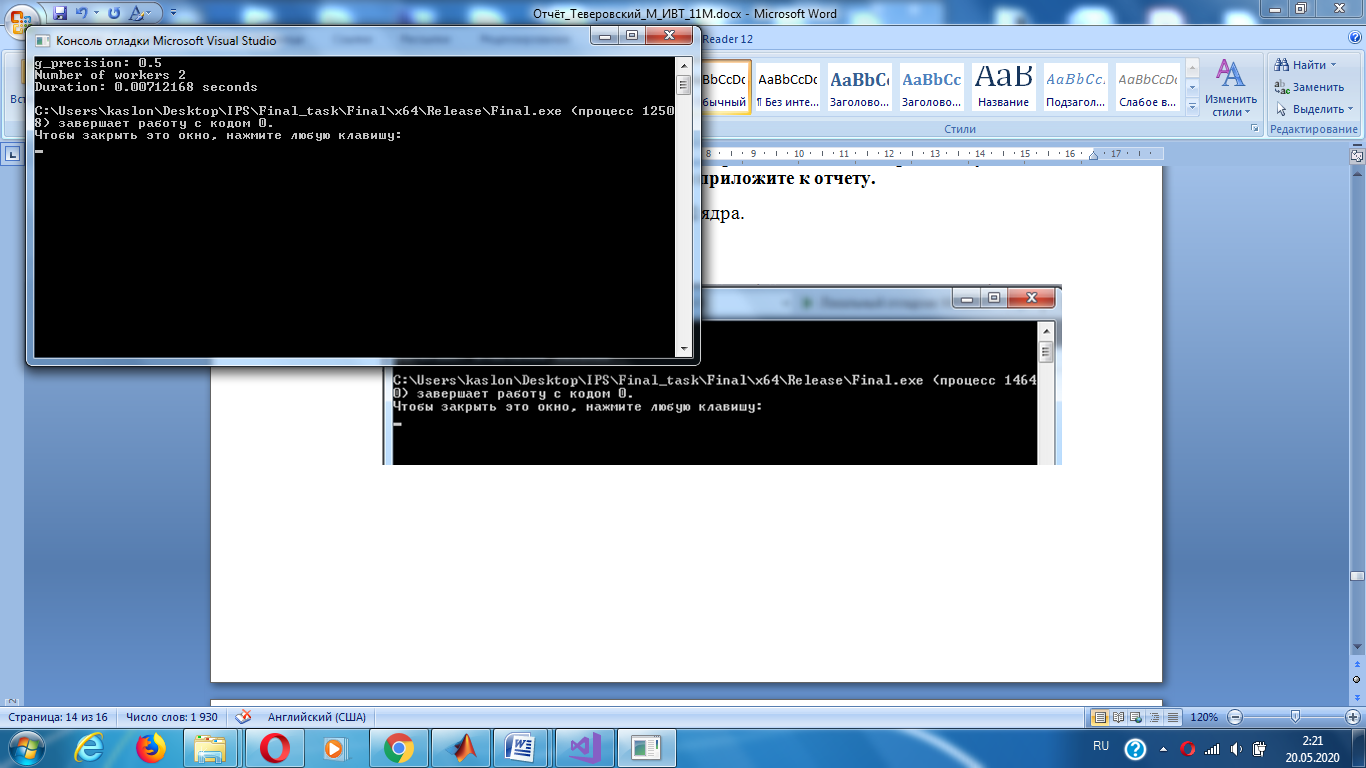
****

Рисунок 15 – Результат при delta = 0.5



Рисунок 16 – Результат при delta = 0.5

3) При delta = 0.1:

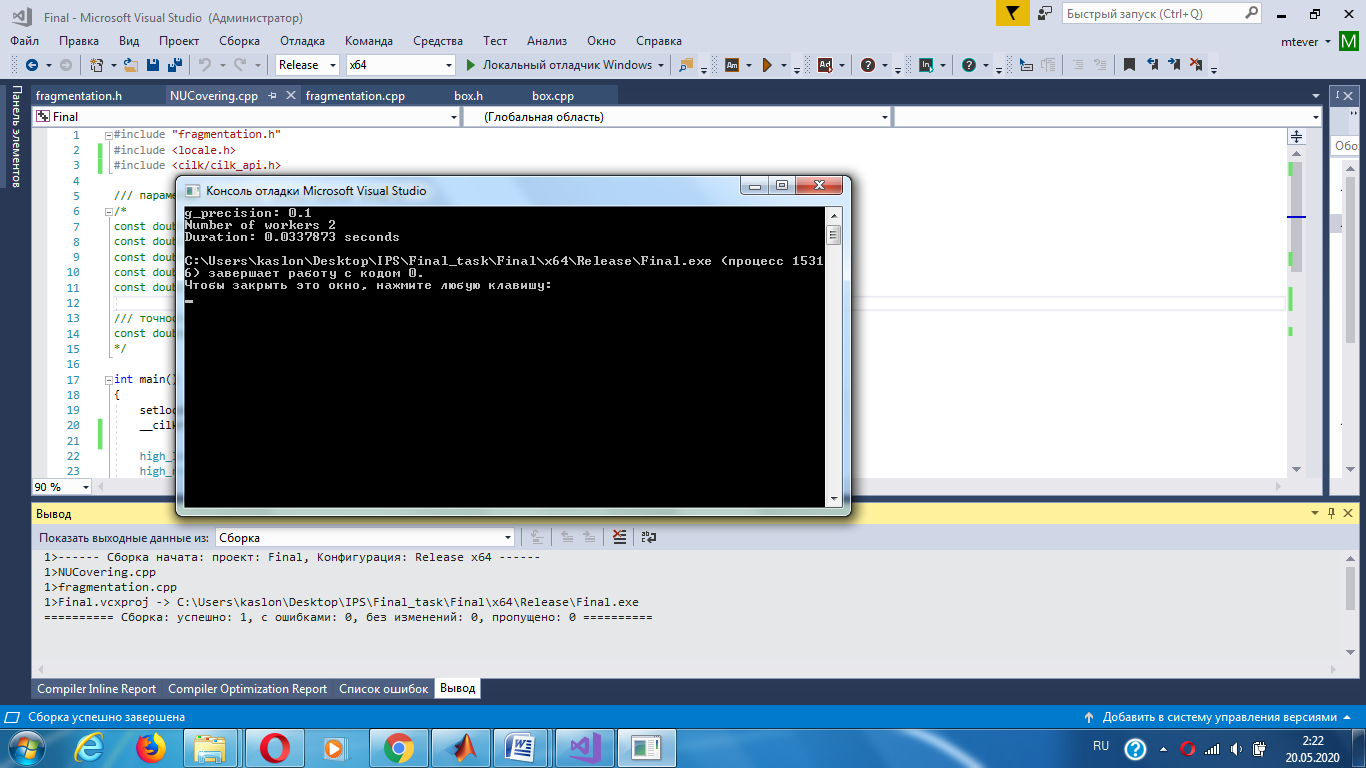
****

Рисунок 17 – Результат при delta = 0.1



Рисунок 18 – Результат при delta = 0.1

|  |  |  |  |
| --- | --- | --- | --- |
| delta | 0.25 | 0.5 | 0.1 |
| Количество ядер | 2 | 2 | 2 |
| Время выполнения | 0.0132568 | 0.00712168 | 0.0337873 |

Таблица 2 – Сравнение результатов

Следовательно: чем больше значение параметра delta, тем быстрее выполняется программа.

**Выводы:**

1) Были реализованы необходимые функции для модели работы планарного робота

2) В ходе выполнения задания были протестированы результаты выполнения последовательного выполнения программы на различных версиях и замерялось время выполнения

3) С помощью пакета MATLAB было построено рабочее пространство для последовательной программы

4) С помощью инструментов IPS: Amplifier XE и Survey Analise был проанализирован последовательный код для введения параллелизма

5) В код был введён параллелизм

6) С помощью Amplifier XE и ParallelAdvisor была проверена правильность работы кода с параллельными участками выполнения

7) Было замерено время выполнения программы в зависимости от количества используемых ядер процессора

8) Было замерено время выполнения программы при использовании 2 ядер в зависимости от параметра delta.