1. **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). Сохраните изображение рабочего пространства. Вставьте его в отчет, назвав соответствующим образом.**

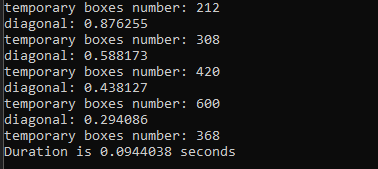


Рис. 1 – Время работы последовательной версии в режиме Debug

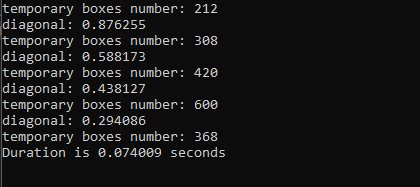


Рис. 2 – Время работы последовательной версии в режиме Release

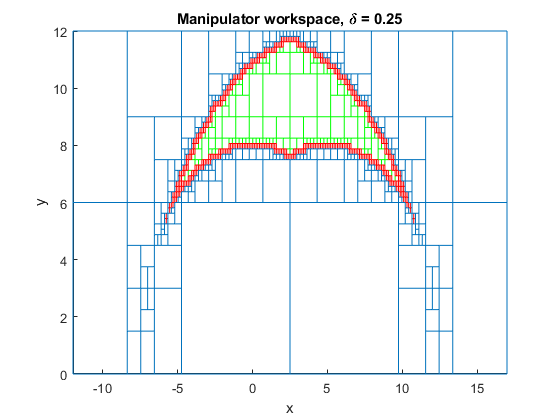


Рис. 3 – Полученное рабочее пространство

1. **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().**

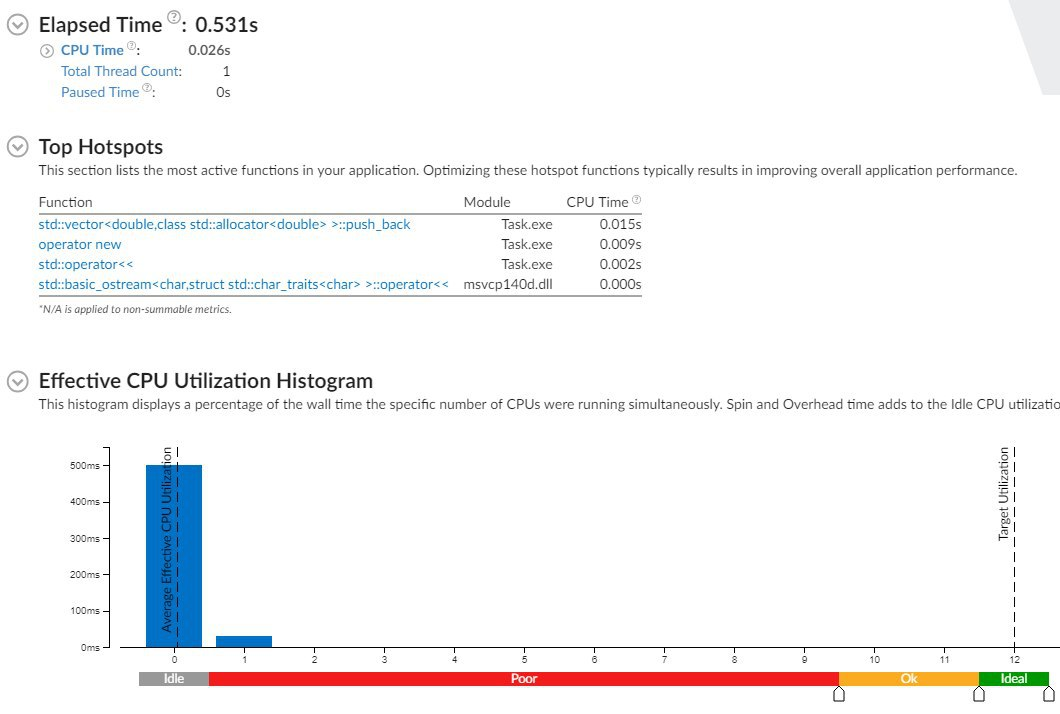


Рис. 4 – Результаты анализа

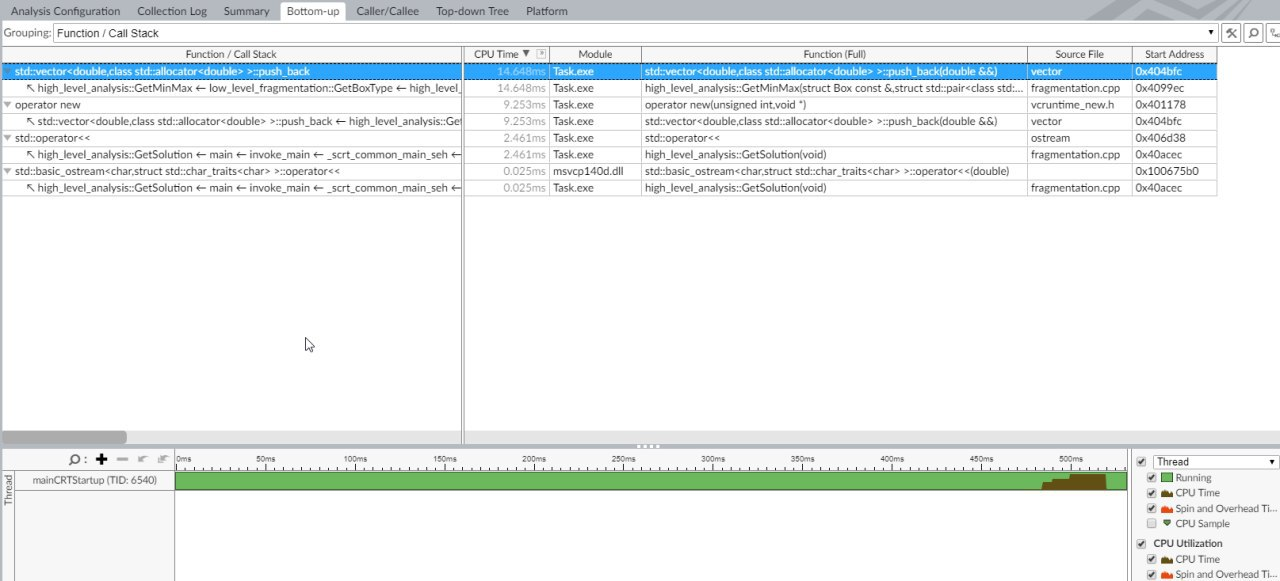


Рис. 5 –Вкладка Bottom-up

1. **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***.**

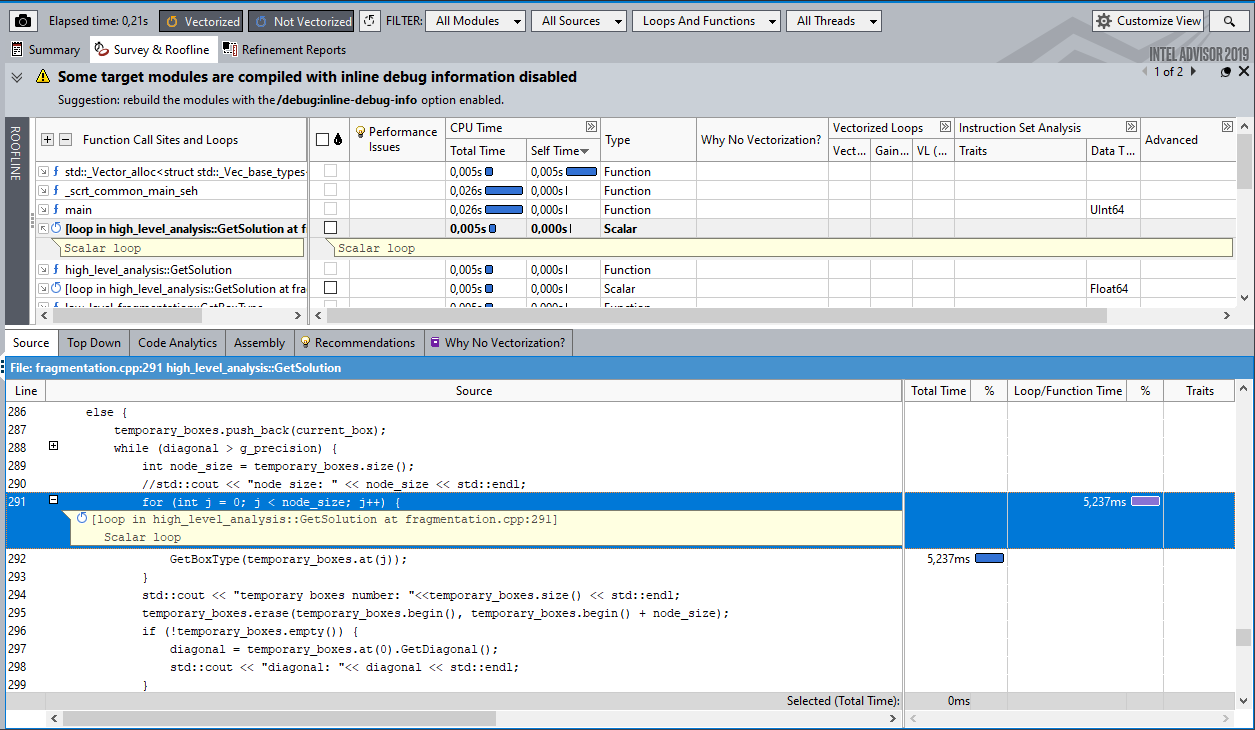


Рис. 6 – Результат Survey Analysis

1. **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.**

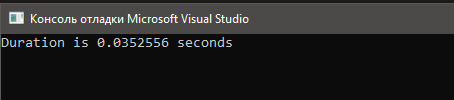


Рис. 7 – Время работы версии с reducer vector в режиме Debug

1. **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.**

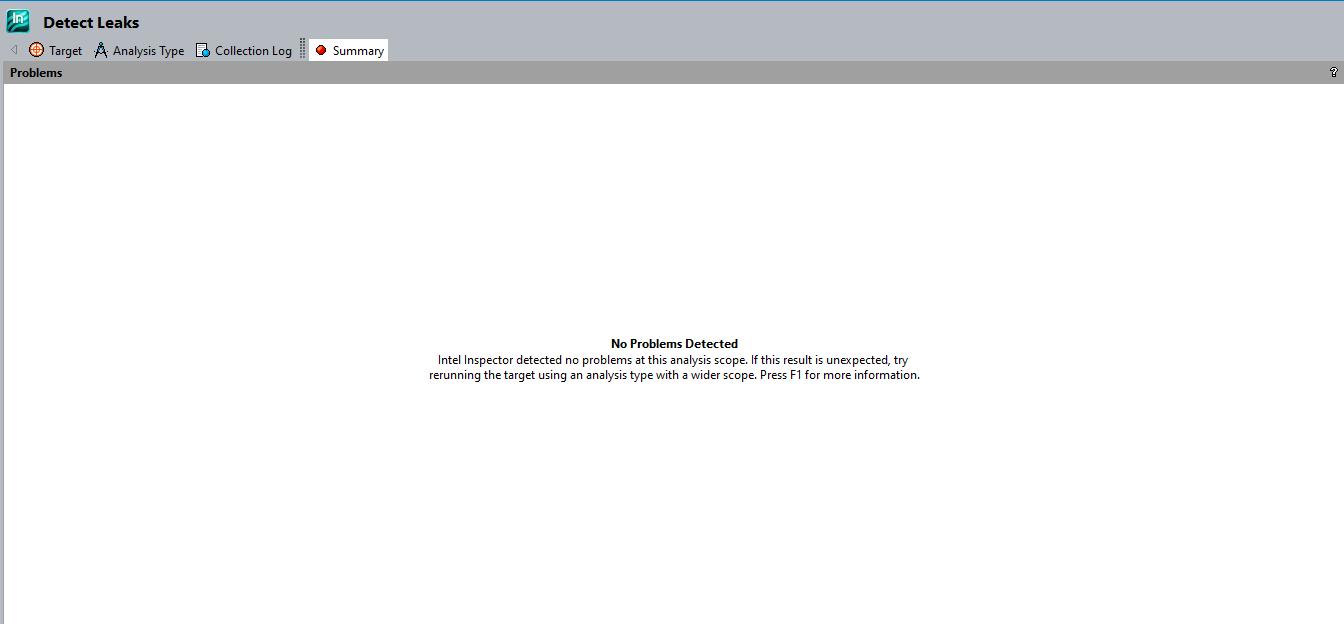
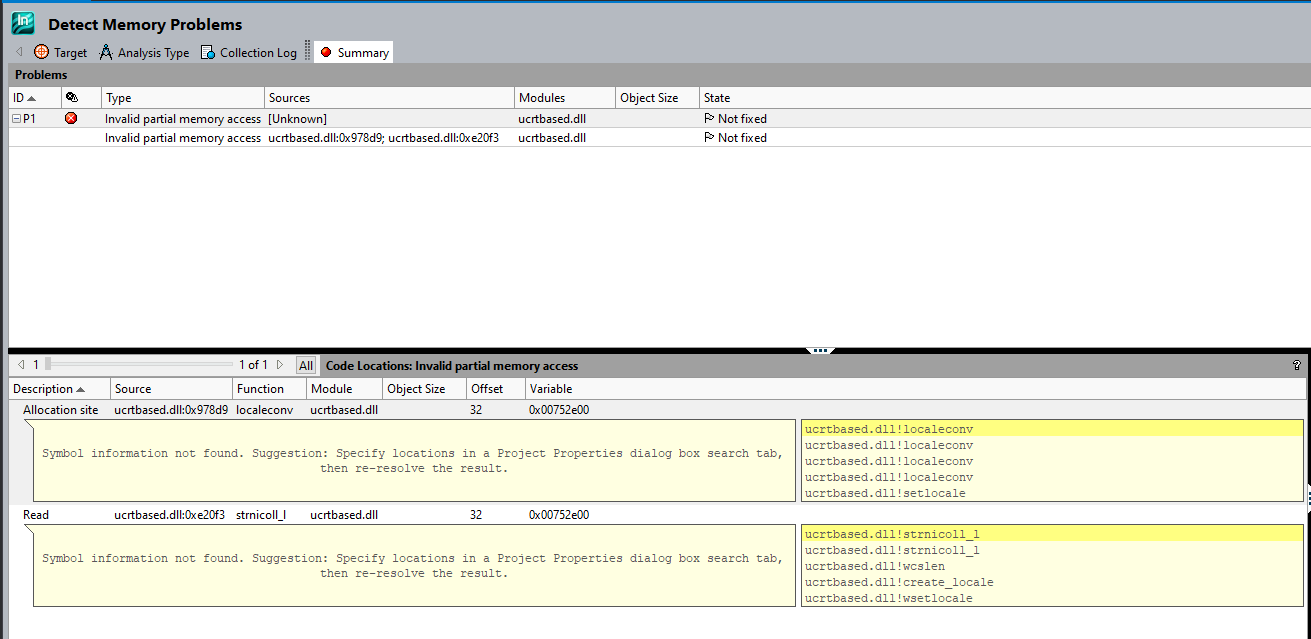


Рис. 8 – Результат Memory Error Analysis Narrowest

Рис. 9 – Результат Memory Error Analysis Medium

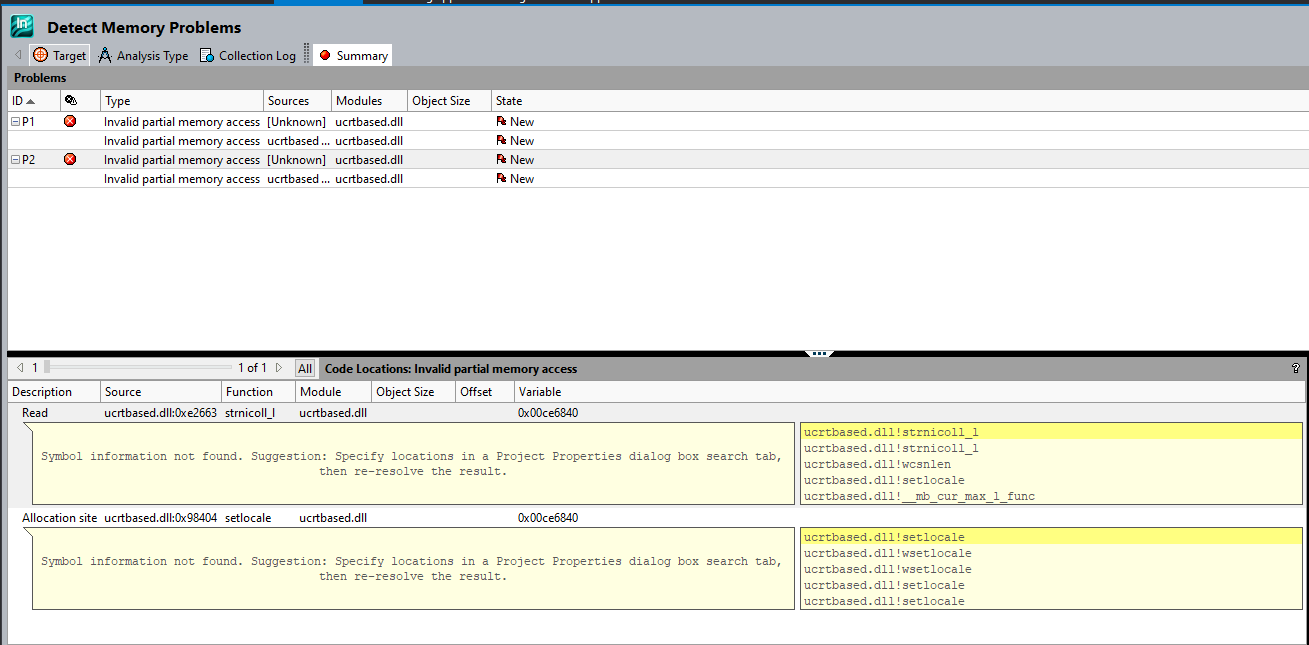


Рис. 10 – Результат Memory Error Analysis Widest

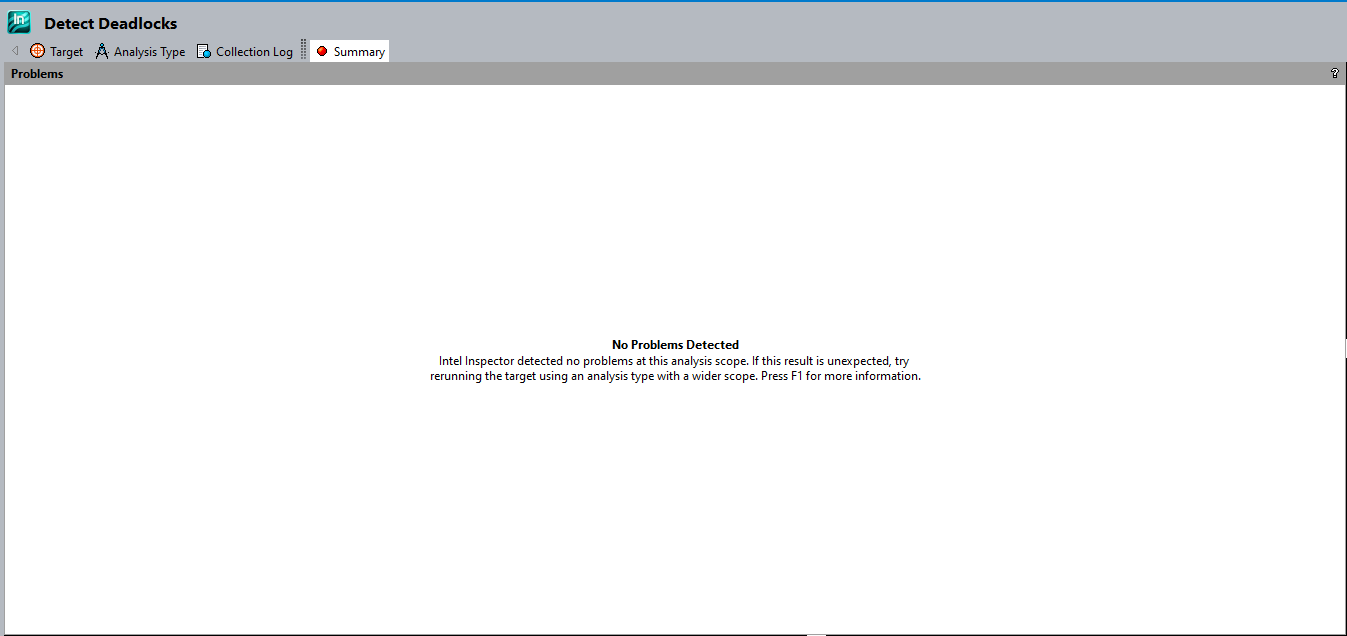


Рис. 11 – Результат Threading Error Analysis Narrowest

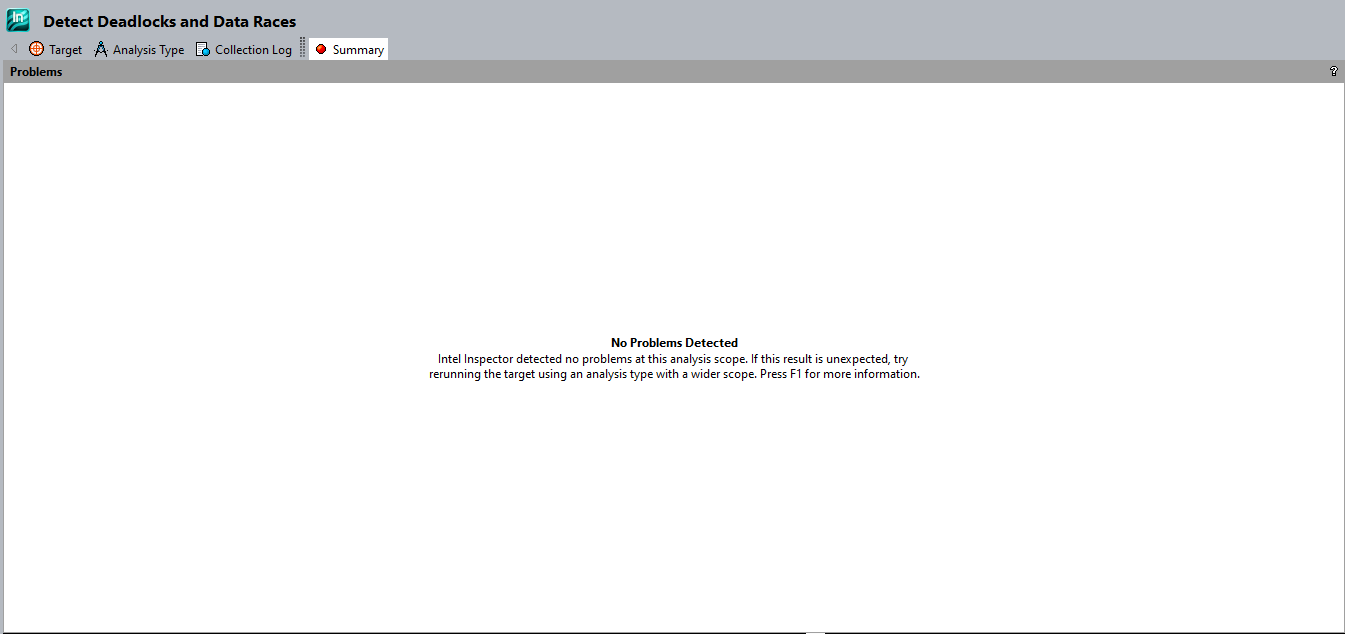


Рис. 12 – Результат Threading Error Analysis Medium

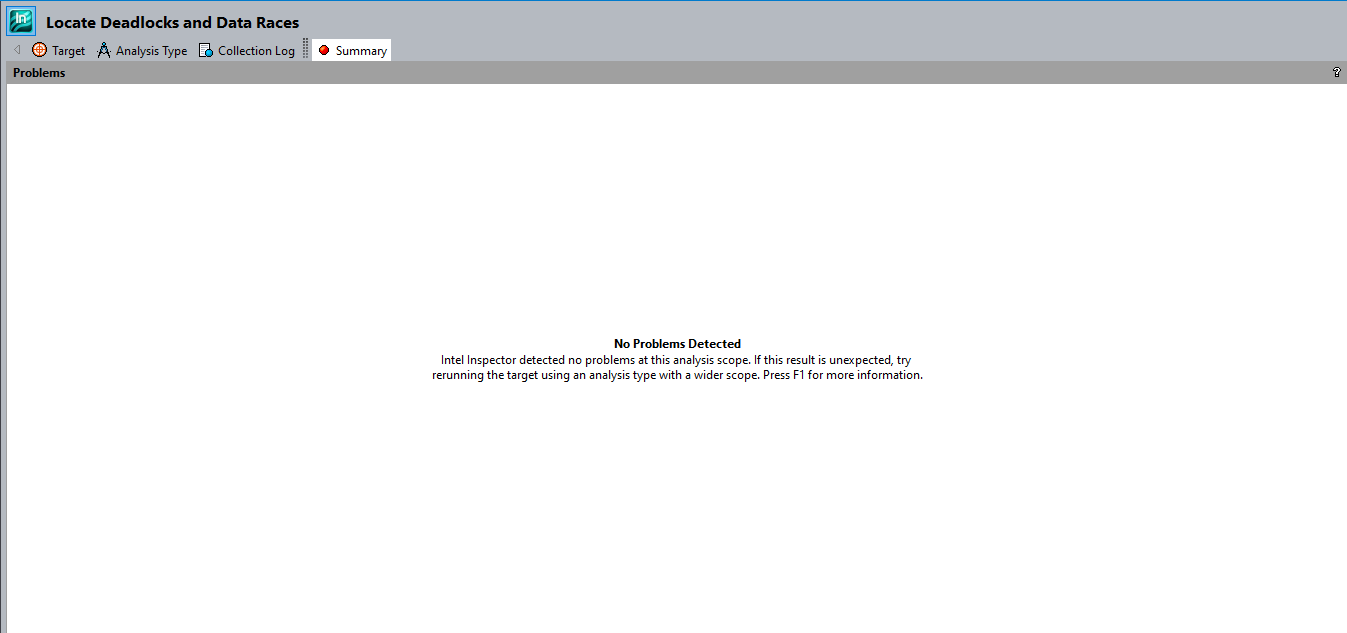


Рис. 13 – Результат Threading Error Analysis Widest

1. **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.**

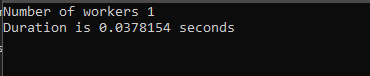


Рис. 14 – Результат с 1 вычислителем

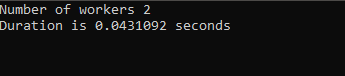


Рис. 15 – Результат с 2 вычислителями

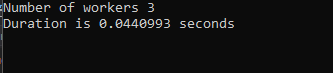


Рис. 16 – Результат с 3 вычислителями

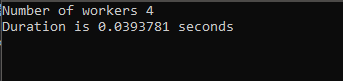


Рис. 17 – Результат с 4 вычислителями

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

const double g\_l1\_max = 20.0;

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

const double g\_l1\_min = 7.0;

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

const double g\_l0 = 5.0;

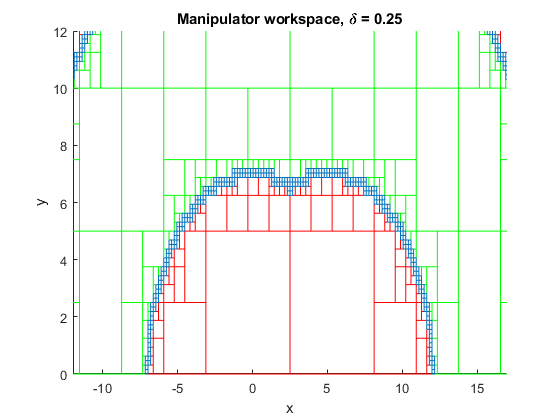
****

Рис. 18 – Рабочее пространство 1

const double g\_l1\_max = 40.0;

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

const double g\_l1\_min = 7.0;

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

const double g\_l0 = 5.0;

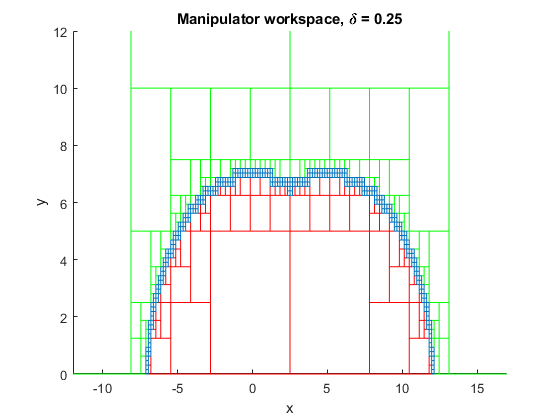
****

Рис. 19 – Рабочее пространство 2

const double g\_l1\_max = 40.0;

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

const double g\_l1\_min = 10.0;

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

const double g\_l0 = 5.0;

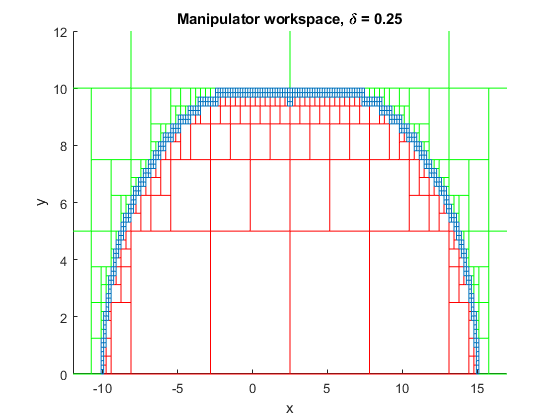
****

Рис. 20 – Рабочее пространство 3