Отчёт по лабораторной работе №5

Жуков Вадим, ИВТ-12М

Технические характеристики

OС: Windows 10

VS: Visual Studio 2017 v15.9.18

IPS: v2019

Железо: 2 физических ядра, 4 логических, 3.4ГГц

# 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" \o "The non-uniform covering approach to manipulator workspace assessment.pdf)

Ознакомился.

# Задание 2

Скачайте следующие файлы: [box.hПредварительный 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".

Определил нужные функции, код находится в файлах проекта.

# 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.

Время работы последовательной версии программы в режиме Debug представлено на рисунке 1, в режиме Release на рисунке 2, полученное рабочее пространство на рисунке 3.

C:\Users\zhuko\OneDrive\Рабочий стол\1.jpg

Рисунок 1. Задание 3, режим Debug

C:\Users\zhuko\OneDrive\Рабочий стол\2.jpg

Рисунок 2. Задание 3, режим Release

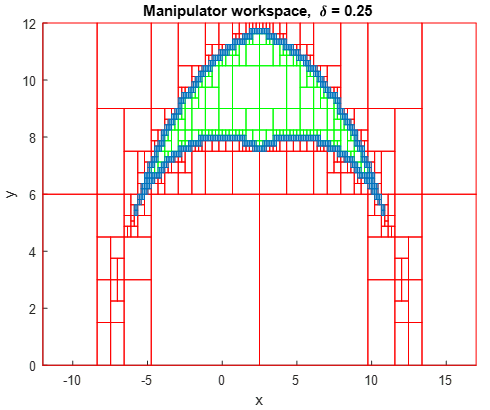


Рисунок 3. Задание 3, рабочее пространство

# 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()***.

Результаты анализа программы в ***Amplifier XE*** представлены на рисунке 4 и на рисунке 5. Функция ***GetMinMax()* действительно присутствует,** на рисунке 5**. Строки кода, отвечающие за запись результатов, пришлось оставить, потому что программа выполняется слишком быстро.**

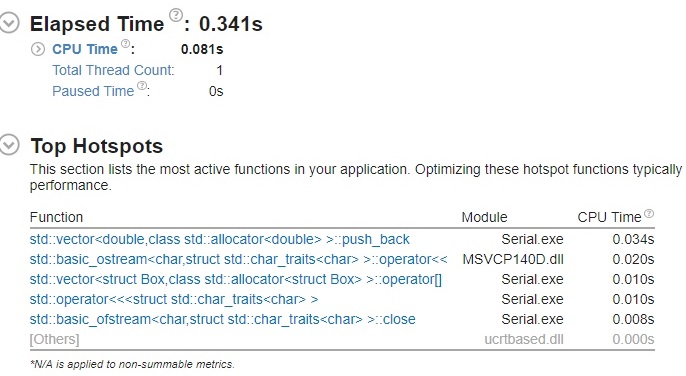


Рисунок 4. Задание 4, вкладка summary

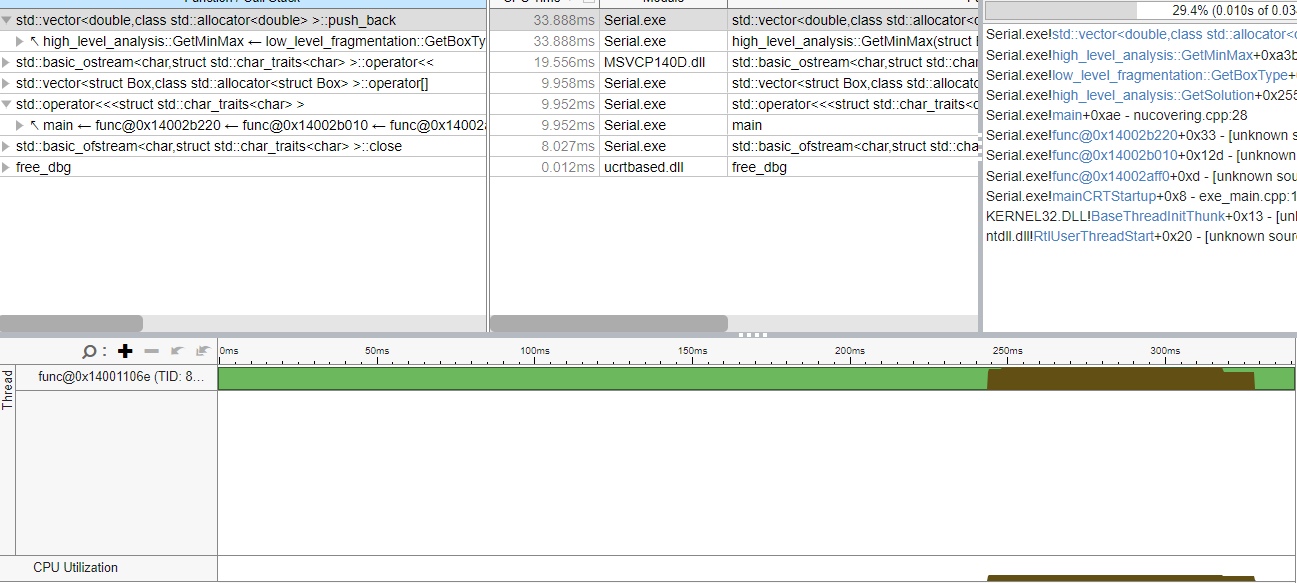


Рисунок 5. Задание 4, вкладка bottom-up

# 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 C\C++ 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**.

Скрины результатов работы ***Survey Analysis*** представлены на рисунке 6, рисунке 7.

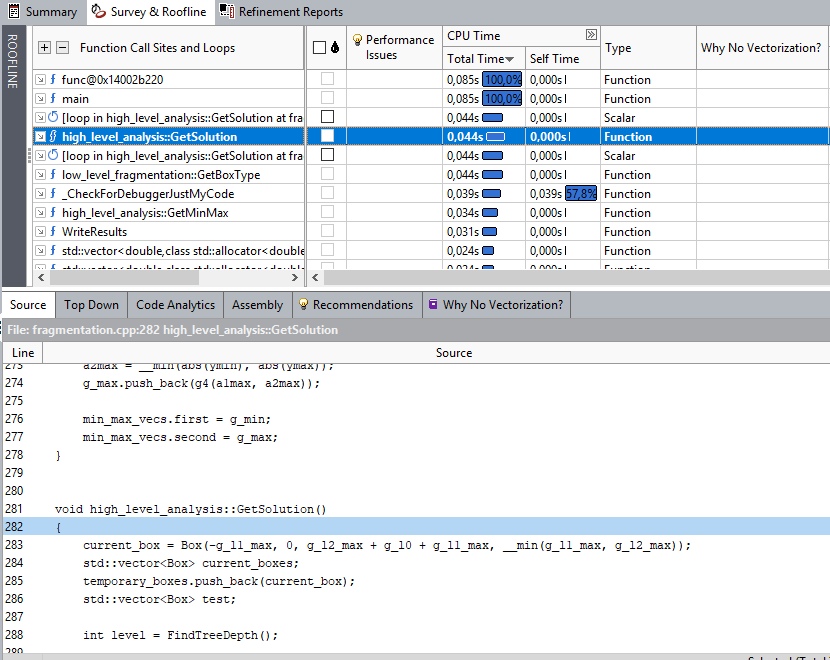


Рисунок 6. Задание 5, скрин 1

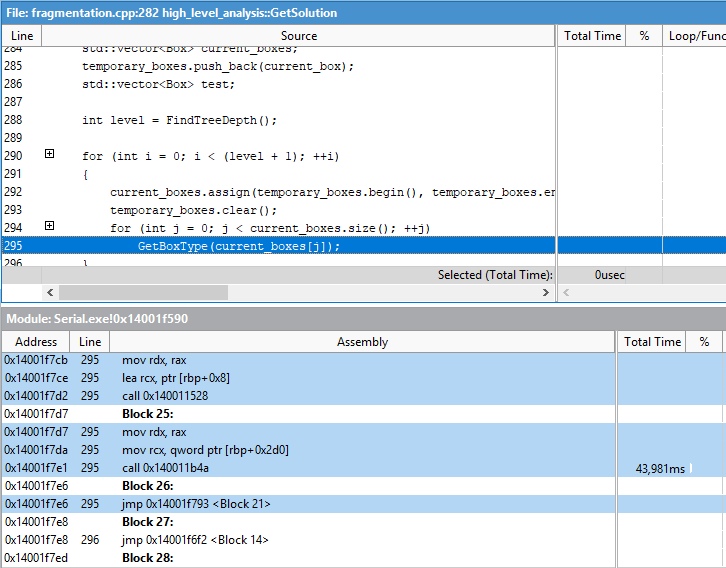


Рисунок 7. Задание 5, скрин 2

# 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.

Результат работы после введения параллелизма представлен на рисунке 8. Как видно, время работы уменьшилось, то есть программа стала работать быстрее.

C:\Users\zhuko\OneDrive\Рабочий стол\8.jpg

Рисунок 8. Задание 6, время работы

# 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 отчёту скрины результатов запуска перечисленных анализов. Исправьте обнаруженные ошибки, приложите скрины результатов анализов, в которых ошибки отсутствуют. *Примечание:* "глюки" ***Intel Cilk Plus*** исправлять не нужно.

Результаты работы ***Inspector XE*** представлены на рисунке 9, рисунке 10 и на рисунке 11. В ходе анализа были обнаружены некоторые ошибки, но они связаны с библиотекой ***cilk*** и ***reducer***, поэтому исправить их не удастся.

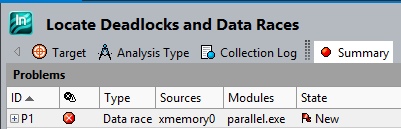


Рисунок 9. Задание 7, скрин 1

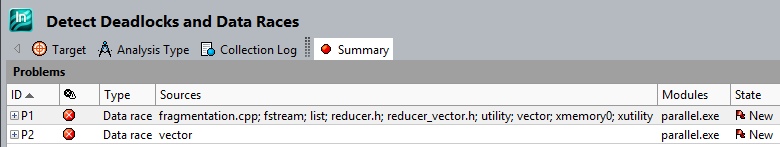


Рисунок 10. Задание 7, скрин 2

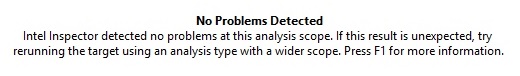


Рисунок 11. Задание 7, скрин 3

# Задание 8

Работа с **Cilk API**. По умолчанию параллельная программа, использующая *Cilk* запускается на количестве поток 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.

Так как на используемом ПК всего 2 ядра, то возможно всего 2 варианта работы программы: на 1 и на 2 ядрах. Результаты представлены на рисунках: рисунок 12 и рисунок 13. Как видно, программа, работающая на 2 ядрах, выполняется быстрее.

C:\Users\zhuko\OneDrive\Рабочий стол\13.jpg

Рисунок 12. Задание 8, результат работы на 1 ядре

C:\Users\zhuko\OneDrive\Рабочий стол\12.jpg

Рисунок 13. Задание 8, результат работы на 2 ядрах

# Задание 9

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

Результаты экспериментов представлены на рисунках ниже. Если эти данные сравнить с теми, что были получены в начале, то видно, что при уменьшении δ время работы программы увеличивается, что ожидаемо, потому что большая точность всегда сопровождается большими затратами.

“Чем больше сила, тем больше и ответственность” ☺

**δ=0.1:**

C:\Users\zhuko\OneDrive\Рабочий стол\14.jpg

Рисунок 14. Время работы при δ=0.1



Рисунок 15. Рабочее пространство при δ=0.1

**δ=0.01:**

C:\Users\zhuko\OneDrive\Рабочий стол\15.jpg

Рисунок 16. Время работы при δ=0.01



Рисунок 17. Рабочее пространство при δ=0.01

**δ=0.25, g\_l1\_max = 0.21:**

C:\Users\zhuko\OneDrive\Рабочий стол\17.jpg

Рисунок 18. Время работы при δ=0.25 и g\_l1\_max = 0.21

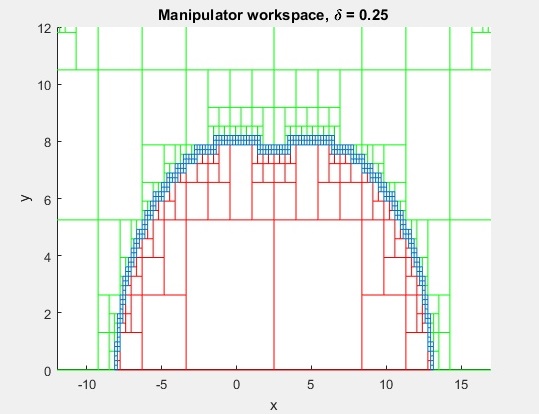


Рисунок 19. Рабочее пространство при δ=0.25 и g\_l1\_max = 0.21