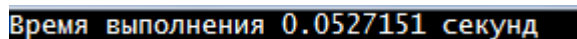


Проектное задание
Выполнил: Селиверстов Михаил
Группа: ИВТ-12М

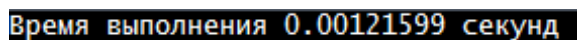
Все необходимые программные коды к рисункам находятся на GitHub:
https://github.com/MichilMIET/final_IPS.git

1. Ознакомьтесь со статьей [The non-uniform covering approach to manipulator workspace assessment.pdf](#).
2. Скачайте следующие файлы: [box.h](#), [box.cpp](#), [fragmentation.h](#), [fragmentation.cpp](#), [NUCovering.cpp](#). В этих файлах представлен предлагаемый каркас разрабатываемого проекта. Ознакомьтесь с содержимым каждого файла. После выполнения **n.1.** Вашей задачей является написание определений тех функций проекта, в теле которых представлен комментарий *"// необходимо определить функцию"*.
3. Реализация последовательной версии программы, определяющей рабочее пространство планарного робота, по предложенному в статье из **n.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**. Сохраните изображение рабочего пространства. Вставьте его в отчет, назвав соответствующим образом.



Время выполнения 0.0527151 секунд

Рисунок 1 - Результат работы программы в режиме Debug



Время выполнения 0.00121599 секунд

Рисунок 2 - Результат работы программы в режиме Release

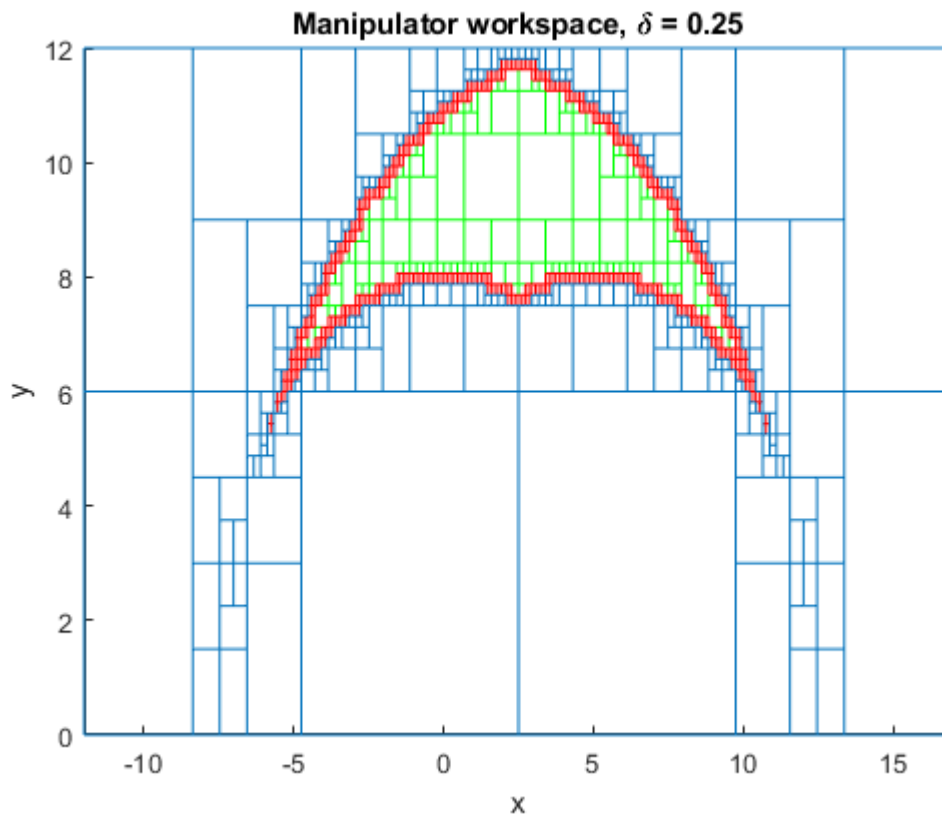


Рисунок 3 - Результат выполнения скрипта PrintWorkspace

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()**.

Elapsed Time[Ⓢ]: 2.055s

CPU Time[Ⓢ]: 0.059s
 Effective Time[Ⓢ]: 0.049s
 Spin Time[Ⓢ]: 0.010s 🚩
 Overhead Time[Ⓢ]: 0s
 Total Thread Count: 2
 Paused Time[Ⓢ]: 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time [Ⓢ]
func@0x10062b1e	ucrtbased.dll	0.030s 🚩
std::vector<double,class std::allocator<double> >::_Getal	serial_IPS.exe	0.010s
std::vector<double,class std::allocator<double> >::assign<double *,0>	serial_IPS.exe	0.010s
std::vector<double,class std::allocator<double> >::_Change_array	serial_IPS.exe	0.010s

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

Рисунок 4 - Результат работы Amplifier XE, вкладка Summary

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.



Рисунок 8 - Результат анализа Memory Error Analysis в режиме Narrowest

ID	Type	Sources	Modules	Object Size	State
P1	Mismatched allocation/deallocation	fragmentation.cpp	labaa.exe		New
P2	Mismatched allocation/deallocation	[Unknown]	nvd3d9wrap.dll		Not fixed
P3	Mismatched allocation/deallocation	[Unknown]	ucrtbase.dll		New
P4	Mismatched allocation/deallocation	xiosbase	labaa.exe		New
P5	Mismatched allocation/deallocation	xlocinfo	labaa.exe		New
P6	Mismatched allocation/deallocation	xlocinfo	labaa.exe		New
P7	Kernel resource leak	[Unknown]	nvinit.dll		Not fixed
P8	Memory leak	[Unknown]	ucrtbase.dll	184	Not fixed

Рисунок 9 - Результат анализа Memory Error Analysis в режиме Medium

Problems						
ID	Type	Sources	Modules	Object Size	State	
P1	Mismatched allocation/deallocation	exe_common.inl	labaa.exe		New	
P2	Mismatched allocation/deallocation	fragmentation.cpp	labaa.exe		New	
P3	Mismatched allocation/deallocation	xiosbase	labaa.exe		New	
P4	Mismatched allocation/deallocation	xlocinfo	labaa.exe		New	
P5	Mismatched allocation/deallocation	xlocinfo	labaa.exe		New	
P6	Kernel resource leak	[Unknown]	nvinit.dll		New	
P7	Memory leak	fragmentation.cpp	labaa.exe	184	New	

Рисунок 10 - Результат анализа Memory Error Analysis в режиме Widest

Detect Deadlocks
Target Analysis Type Collection Log Summary

Problems

No Problems Detected

Intel Inspector detected no problems at this analysis scope. If this result is unexpected, try rerunning the target using an analysis type with a wider scope. Press F1 for more information.

Рисунок 11 - Результат анализа Threading Error Analysis в режиме Narrowest

Problems					
ID	Type	Sources	Modules	State	
P1	Data race	list	labaa.exe	New	
P2	Data race	list; reducer.h; utility; vector; xmemory	labaa.exe	New	

Рисунок 12 - Результат анализа Threading Error Analysis в режиме Medium

ID	Type	Sources	Modules	State
P1	Data race	list	labaa.exe	Not fixed
P2	Data race	list; reducer.h; utility; vector; xmemory	labaa.exe	New
P3	Data race	list; xmemory	labaa.exe	New
P4	Data race	list; xmemory	labaa.exe	Not fixed

Рисунок 13 - Результат анализа Threading Error Analysis в режиме Widest

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 заголовочный 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 *X* - 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 *X*, 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.

В моей системе 4 ядра, но у меня используется Hyper-threading поэтому результат не такой однозначный, получается работает медленнее при увеличении количества ядер.

```
Number of workers 1
Время выполнения 0.00262867 секунд
```

Рисунок 14 - Результат работы с 1 ядром

```
Number of workers 2
Время выполнения 0.00654459 секунд
```

Рисунок 15 - Результат работы с 2 ядрами

```
Number of workers 3
Время выполнения 0.00381693 секунд
```

Рисунок 16 - Результат работы с 3 ядрами

```
Number of workers 4
Время выполнения 0.00566566 секунд
```

Рисунок 17 - Результат работы с 4 ядрами

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

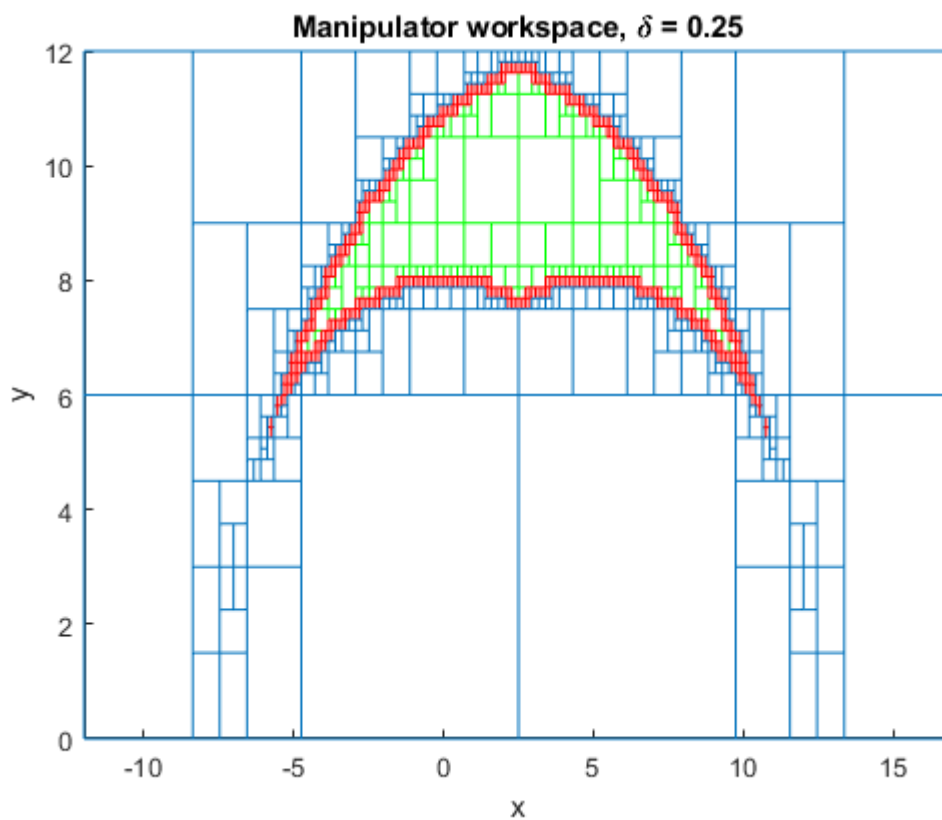


Рисунок 18 - Результат визуализации