**King Abdul Aziz University**

**Faculty of** **Computing and Information Technology**

**Department of Computer Science**

**CP499 Project Progress Report Part 2**

**Title: Approximate Query Location in Mobile Computing**

**(Using GPU)**

**Supervisor: Dr.Jonathan (Yahya) Cazalas**

**Signature:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Member** | **Name** | **ID** | **E-Mail** |
| **1** | **Ahmad Ghassan Hariri** | **1008381** | **Ahmadgjh@gmail.com** |
| **2** | **Abdullah Salem Khered** | **1008623** | **Khered20@gmail.com** |

Table of Contents

[Summary of previous work 4](#_Toc386305632)

[Aim of the project 4](#_Toc386305633)

[Problem definition 4](#_Toc386305634)

[Project scope 4](#_Toc386305635)

[Expected outcome 4](#_Toc386305636)

[Target Users 4](#_Toc386305637)

[Suggested solution 5](#_Toc386305638)

[Functional Requirements :- 5](#_Toc386305639)

[Nonfunctional Requirements :- 5](#_Toc386305640)

[Data Requirements :- 5](#_Toc386305641)

[Hardware :- 5](#_Toc386305642)

[Software :- 5](#_Toc386305643)

[Methods and Approaches 6](#_Toc386305644)

[Planning 6](#_Toc386305645)

[Analysis 6](#_Toc386305646)

[Design 6](#_Toc386305647)

[Implementation 6](#_Toc386305648)

[Maintenance 6](#_Toc386305649)

[Activities and Actions 7](#_Toc386305650)

[Use case diagram :- 7](#_Toc386305651)

[Flow chart :- 8](#_Toc386305652)

[Class diagram :- 9](#_Toc386305653)

[Tasks Completed : 9](#_Toc386305654)

[Implementation 10](#_Toc386305655)

[Tools & technologies 10](#_Toc386305656)

[Interface 11](#_Toc386305657)

[Testing 13](#_Toc386305658)

[References 34](#_Toc386305659)

[Research papers : 34](#_Toc386305660)

[Book : 34](#_Toc386305661)

[Code 35](#_Toc386305662)

# Summary of previous work

## Aim of the project

The aim is to learn parallel programming on GPUs using NVidia’s CUDA and use this knowledge to solve problems & improve the evaluations in the domain of real-time approximate query location in mobile computing.

## Problem definition

The problem is about a big number of objects and those objects are moving randomly. All those objects send their locations in real-time to the server. Also, those objects want to know where their neighbors .

We tackle a challenge in processing continuous range queries: coping with data uncertainty inherently associated with location data (stale, inaccurate position).

These queries must be evaluated in REAL-TIME. Parallel programming on the GPU provide a great solution to evaluate these queries.

## Project scope

The scope of this project is:

1. Learn CUDA C.
2. Make simulations in the GPU & CPU to solve approximate queries.
3. Improve the evaluation for the approximate query locations.
4. Compare the time results between the CPU & the GPU.

## Expected outcome

CUDA program that parallelize the execution of large number of queries for finding ( the neighbors ) with approximate locations.

## Target Users

The target users are large environments in which needs to be improved by querying to find approximate location to theirs neighbors in order to achieve their goals (avoid causing accidents, find, manage, manipulate,... etc.).

## Suggested solution

First, we will create four simulations in CPU & GPU to model the environment – movement- by giving a coordinate to every object and letting it move randomly.

In the CPU we will make two simulations:

* Simulation to solve range queries using exact locations.
* Simulation to solve range queries using approximate locations.

Also, In the GPU we will make the same simulations to improve the evaluation.

## Functional Requirements :-

* Simulate an environment with continuous moving objects.
* Find exact & approximate locations by an algorithm.
* Compare & evaluate the performance between the CPU & the GPU.

## Nonfunctional Requirements :-

* Response time.
* Throughput.
* Reliability.
* Maintainability.

## Data Requirements :-

The xCoordinate & yCoordinate for every object and the query range.

## Hardware :-

GPU Nvidia card supported CUDA.

## Software :-

* Microsoft visual studio ( C Language ) .
* Windows 8 .
* CUDA SDK 5.5 .
* CUDA Device Driver.
* NVIDIA NSight VISUAL STUDIO Edition Tool .

# Methods and Approaches

## Planning

We planned for the project by work break down structure (WBS) then scheduled the project by Gantt Chart & time estimates to manage & organize our work in order to complete the project.

## Analysis

We started by reading about the problem in research papers and read a book teach us the CUDA language.

Then, We defined the project scope , functional requirements , non-functional requirements and the expected outcome.

## Design

We designed an use case diagram , flow chart and class diagram.

## Implementation

we started coding the project . and the coding is divided into 6 parts :

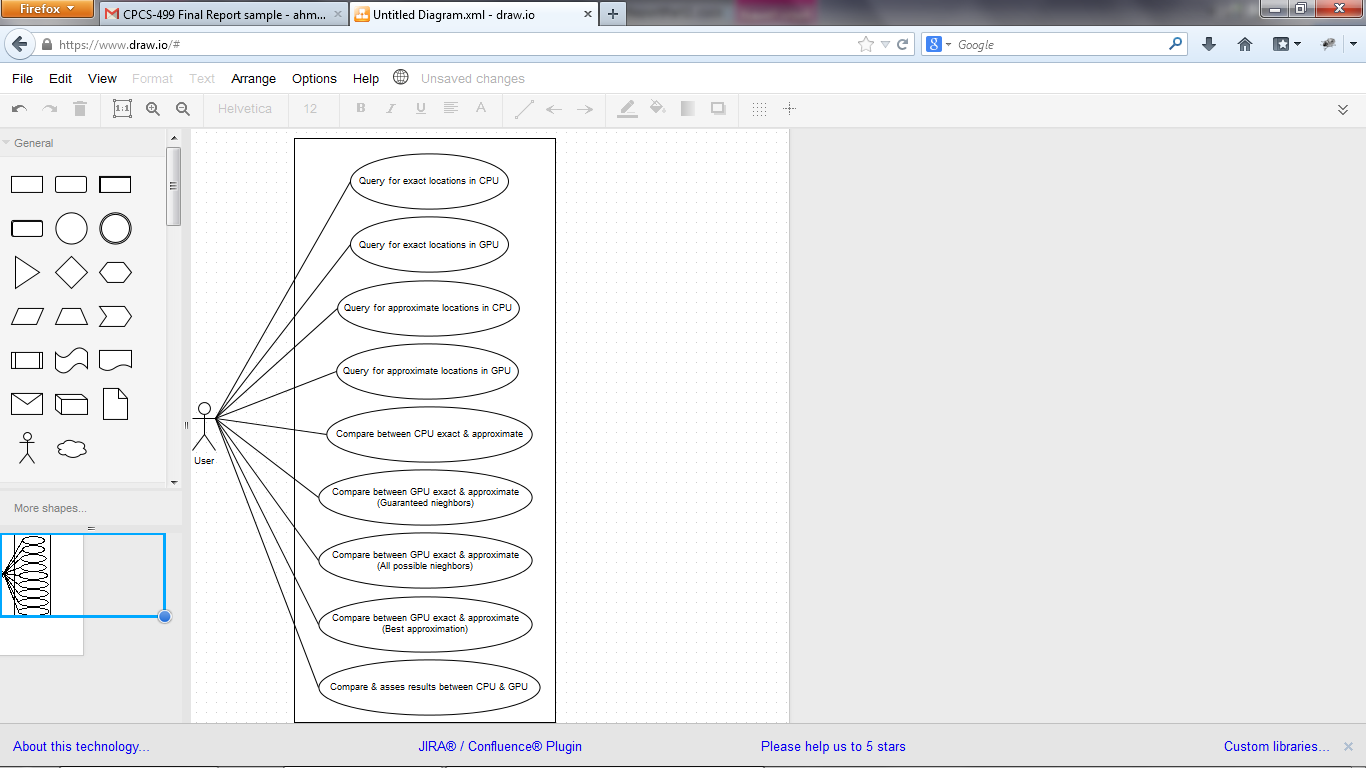
1. Simulate an environment with continuously moving objects.
2. Find exact location by an algorithm in CPU.
3. Find exact location by an algorithm in GPU.
4. Find approximate location by an algorithm in CPU.
5. Find approximate location by an algorithm in CPU.
6. Compare the results between CPU exact and approximate locations.
7. Compare the results between GPU exact and approximate locations (Guaranteed neighbors).
8. Compare the results between GPU exact and approximate locations (All possible neighbors).
9. Compare the results between GPU exact and approximate locations(Best approximation).
10. Compare & asses the results between the CPU & the GPU.

## Maintenance

After completing the code , we will maintain the program by making a lot of tests in order to remove the errors.

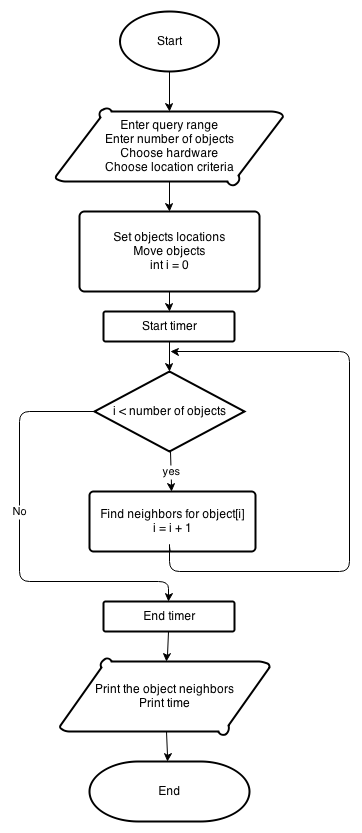
# Activities and Actions

## Use case diagram :-



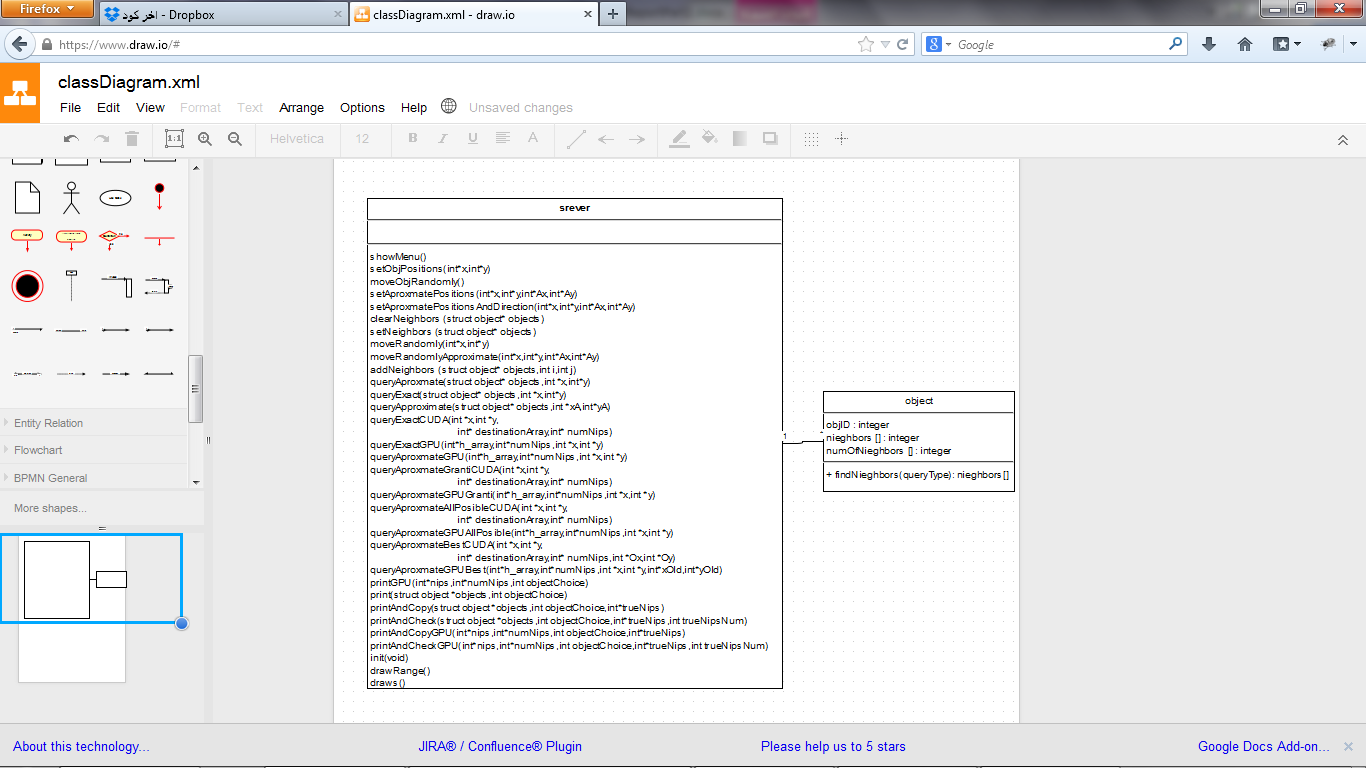
This use case diagram shows what kind of query the user can choose from.

## Flow chart :-

****

This flow chart explains how the system will work.

## Class diagram :-



This class diagram show the role of the server and the role of the client.

## Tasks Completed :

1. Simulate an environment with continuously moving objects.
2. Find exact locations by an algorithm in CPU.
3. Find exact locations by an algorithm in GPU.
4. Find approximate locations by an algorithm in CPU.
5. Find approximate locations by an algorithm in GPU.
6. Compare the results between CPU exact and approximate locations.
7. Compare the results between GPU exact and approximate locations (Guaranteed neighbors).
8. Compare the results between GPU exact and approximate locations (All possible neighbors).
9. Compare the results between GPU exact and approximate locations (Best approximation).
10. Compare & asses the results between the CPU & the GPU.

# Implementation

## Tools & technologies

**Hardware :-**

We need GPU NVidia card supports CUDA in order to do parallel computations.

**Software :-**

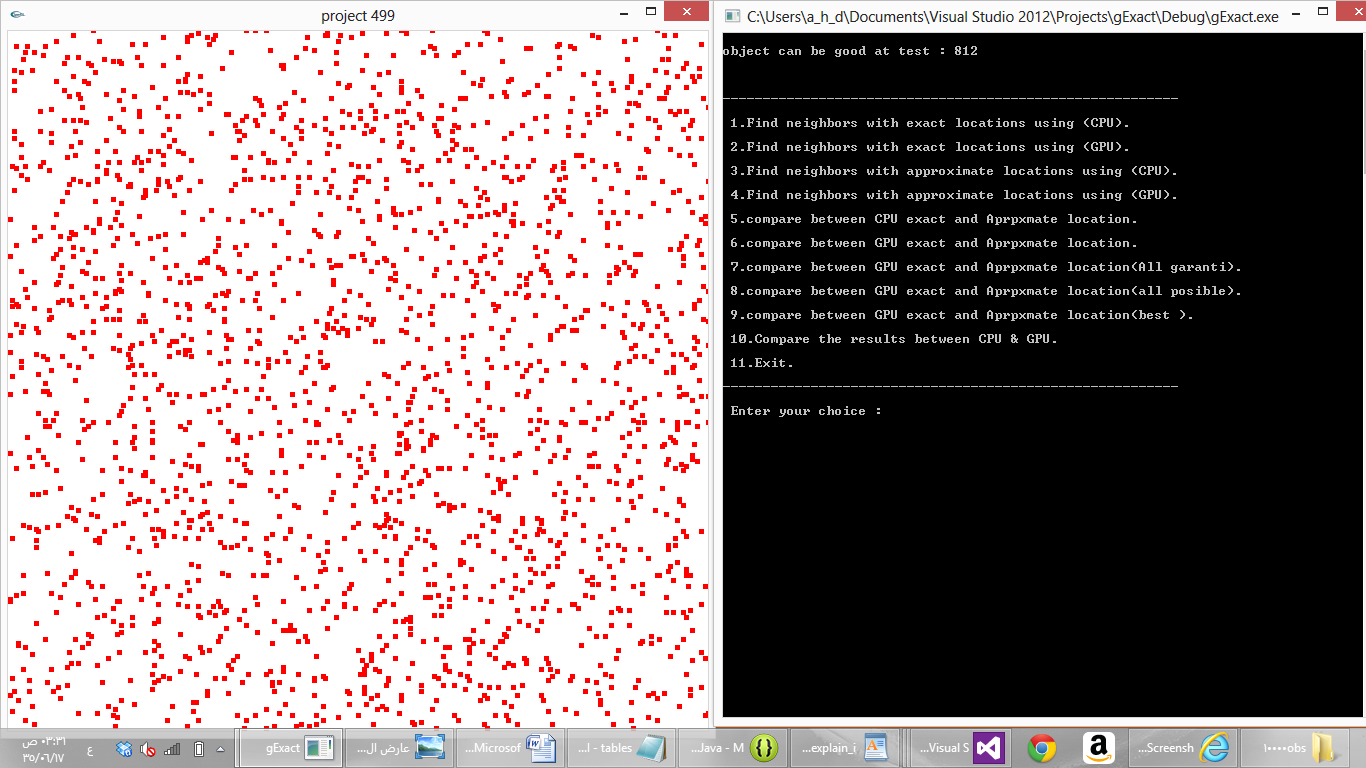
* Microsoft visual studio ( C Language ) , C is an interface to the CUDA platform.
* Windows 8 .
* CUDA SDK 5.5 .
* CUDA Device Driver.
* NVIDIA NSight Visual Studio Edition Tool , it brings parallel programming on the (GPU) into visual studio.
* Microsoft Visio ( To design the project ).

## Interface

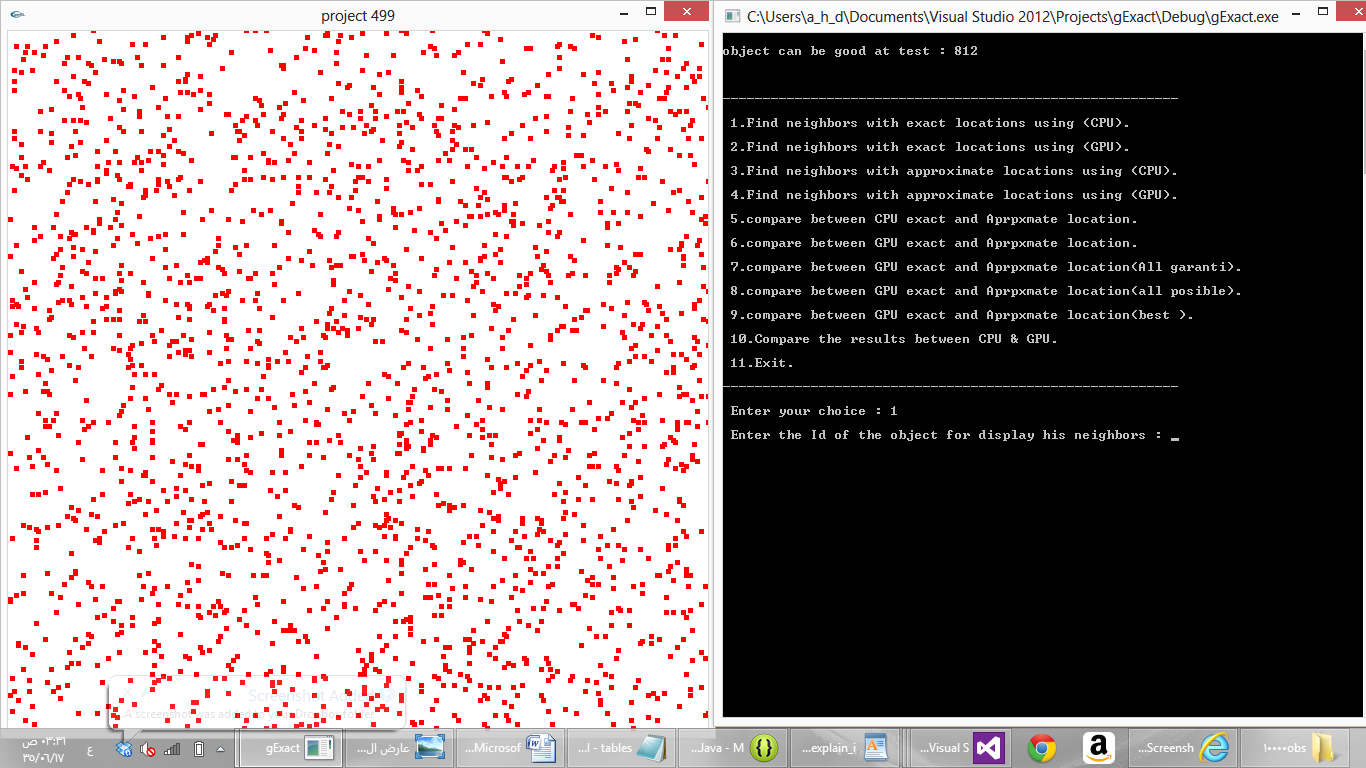
The interface is a command line interface ( because the project is simulation-based type ).

The interface :

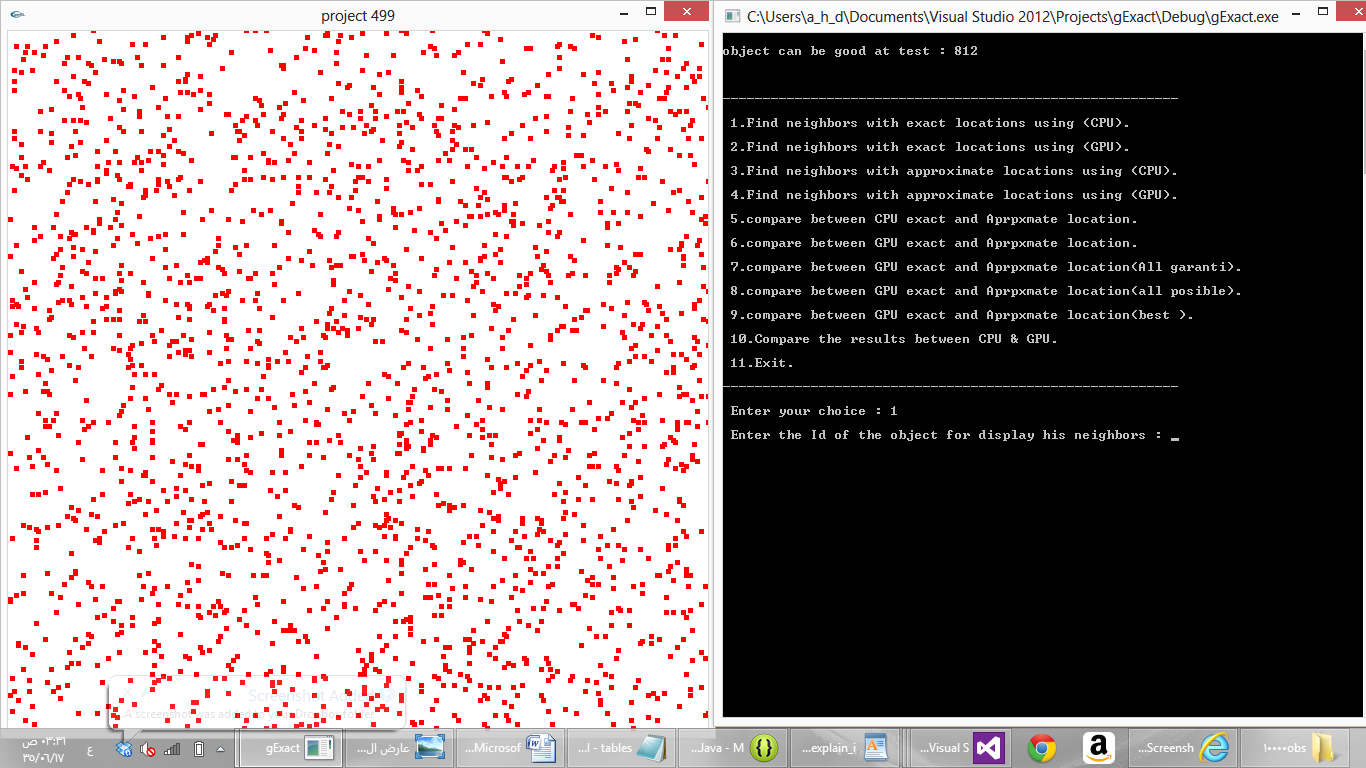
* The user need to enter a choice from the list.



* The system will suggest an object ID , in order to fit the OpenGL screen ( the screen cannot show thousands of objects because the PC screen is small ) so the system will provide a candidate object to find his neighbors.



* The user needs to type the ID of the object in order to display the neighbors.



* After typing an ID the object will marked with a blue color and a blue range around the object will be created.

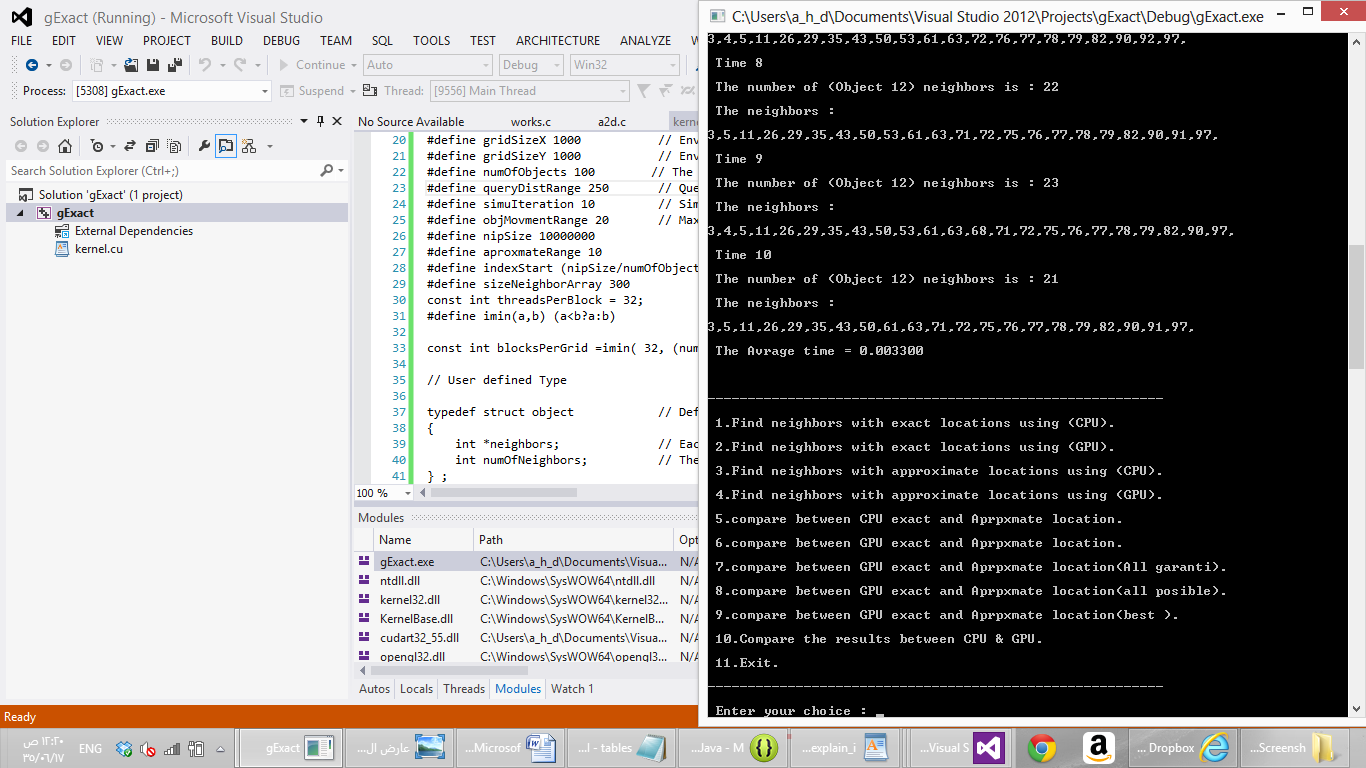
# Testing

We will do a unit test in order to test each part ( Query type ) of the system to check the correctness of the results. We check the correctness of the results by comparing the results between CPU & GPU and by visualizing the simulation in order to see and count the objects that falls within the range.

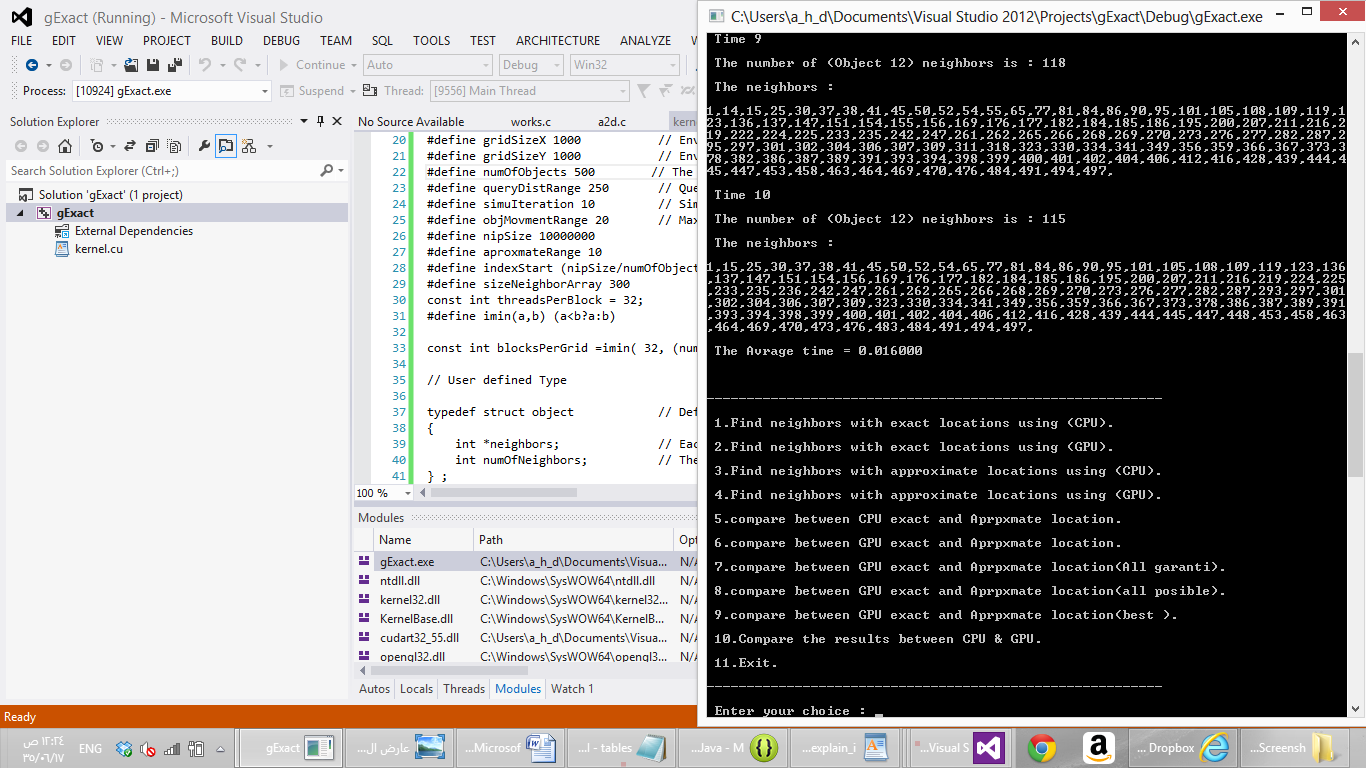
**The test cases :**

* **Location : Exact**

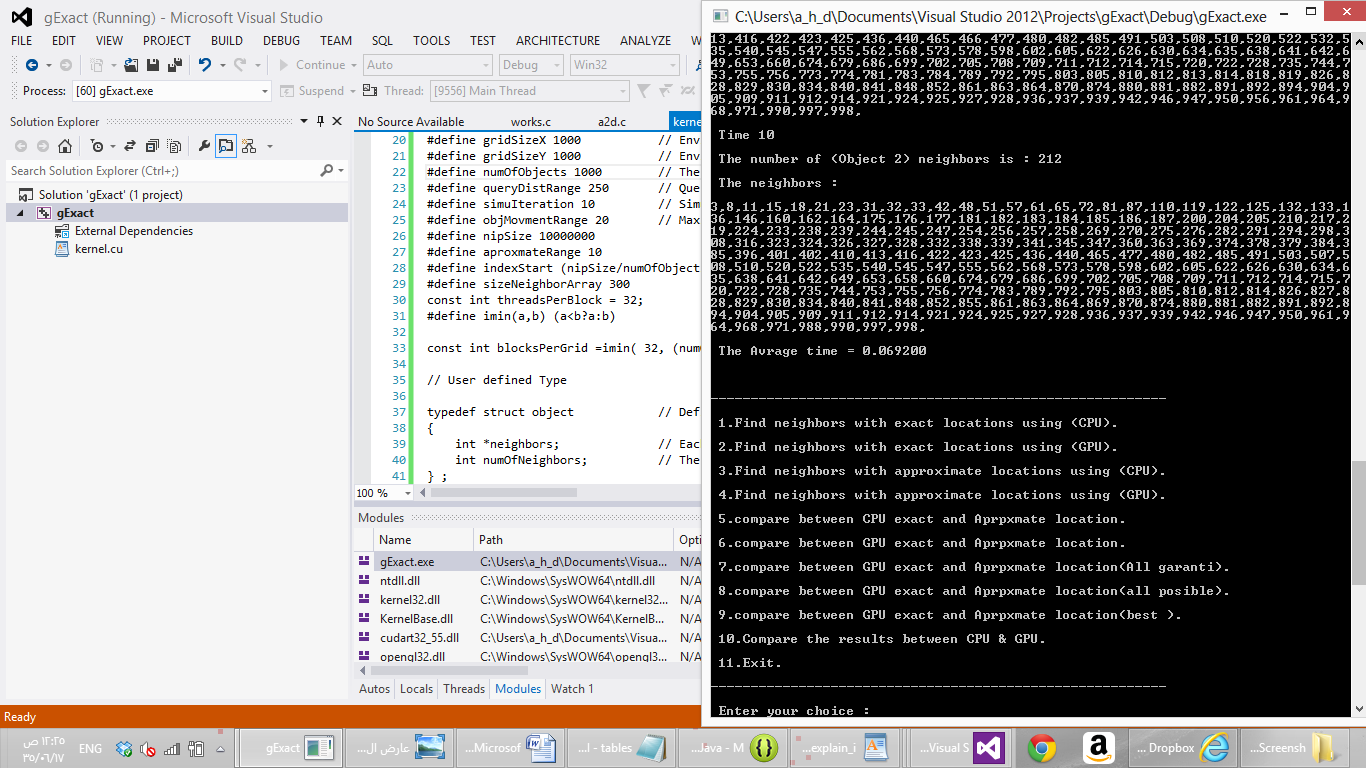
Number of objects = 100 in CPU



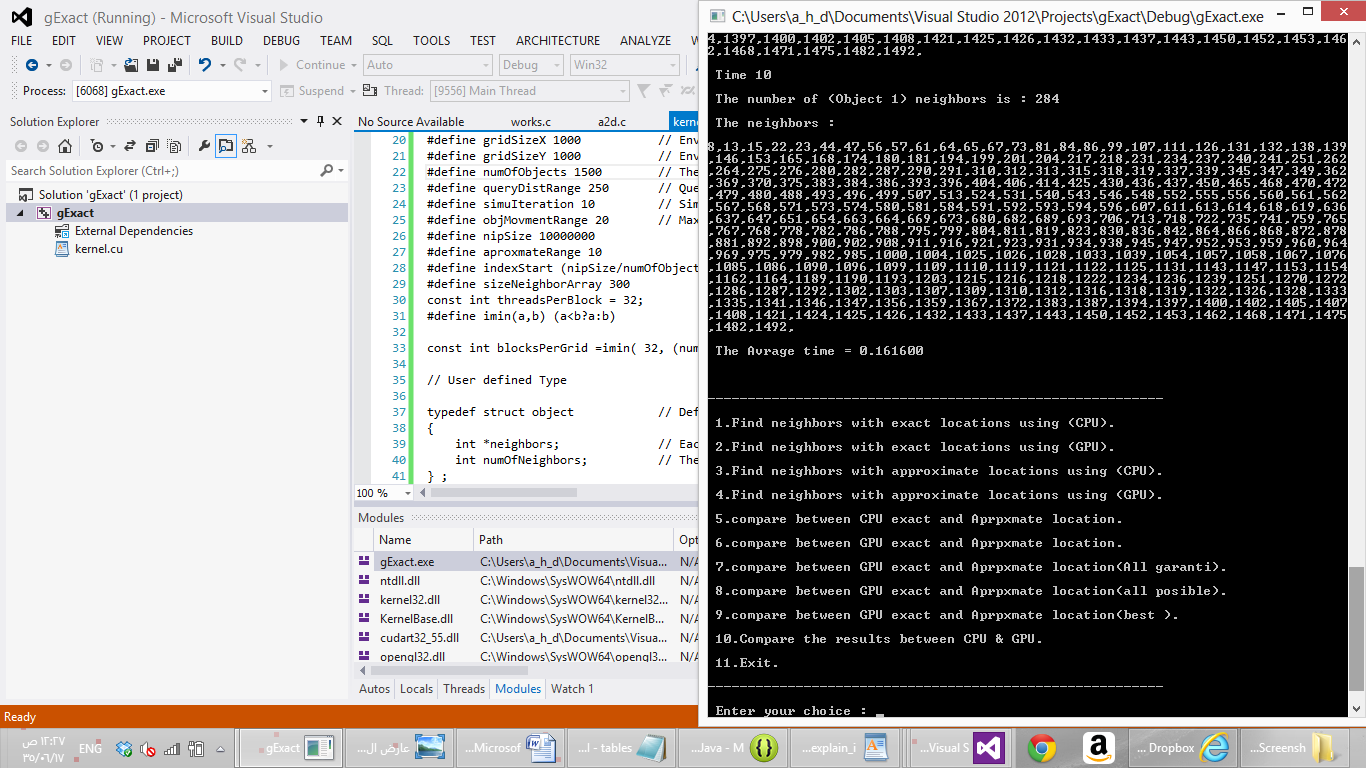
Number of objects = 500 in CPU



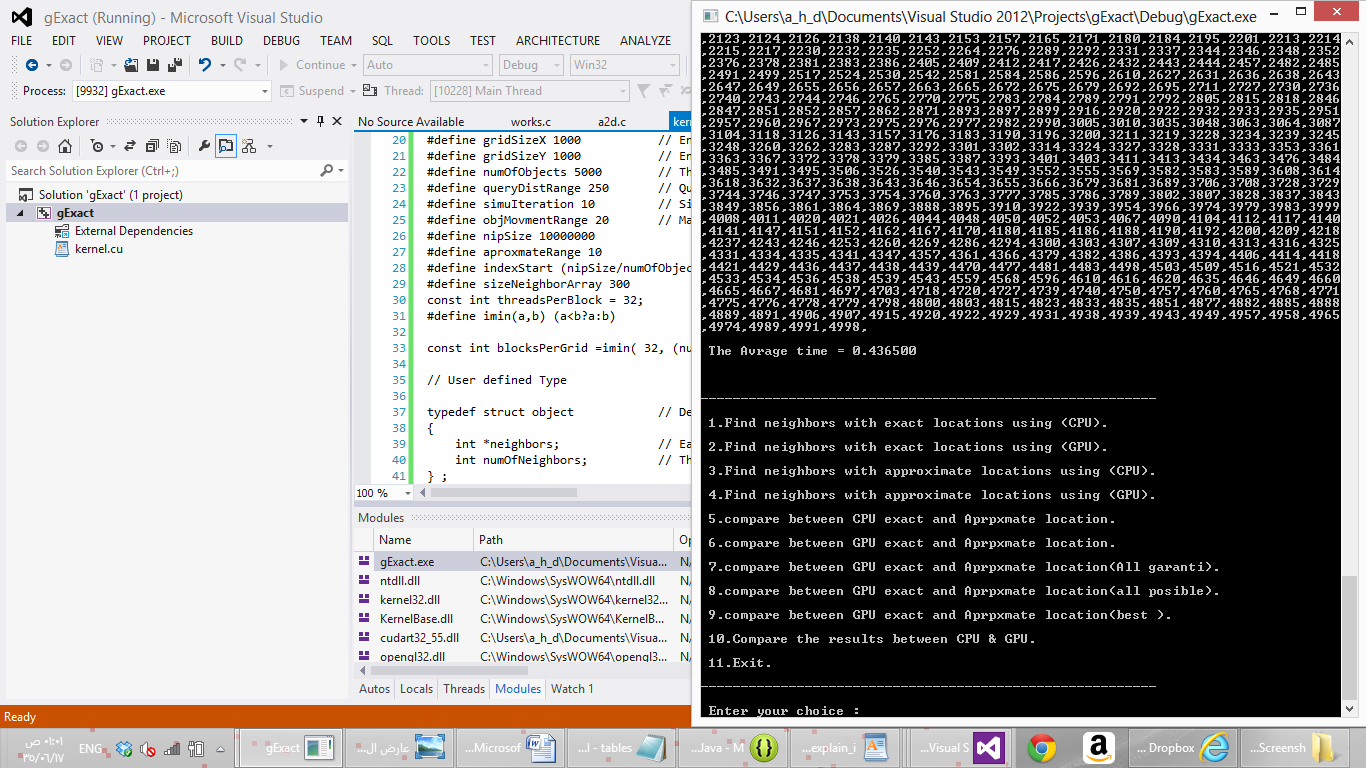
Number of objects = 1000 in CPU



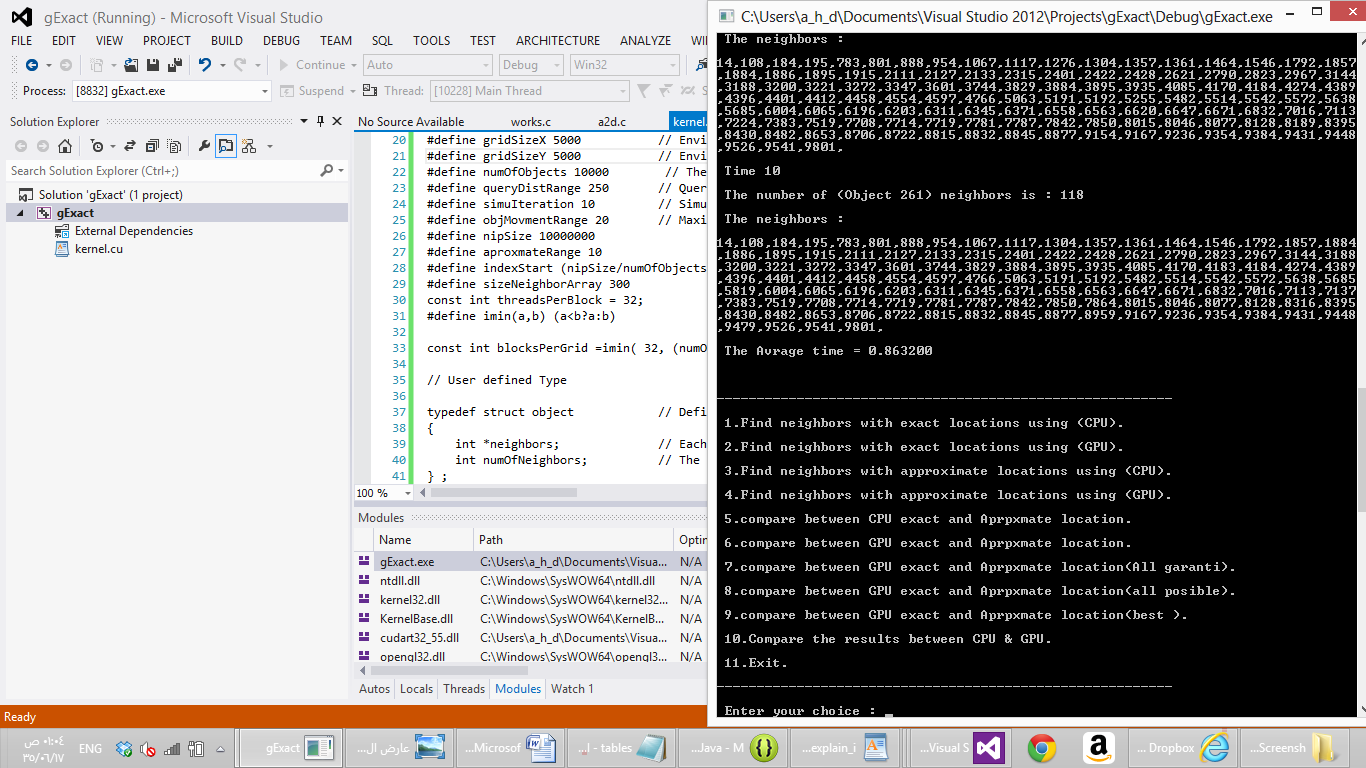
Number of objects = 1500 in CPU



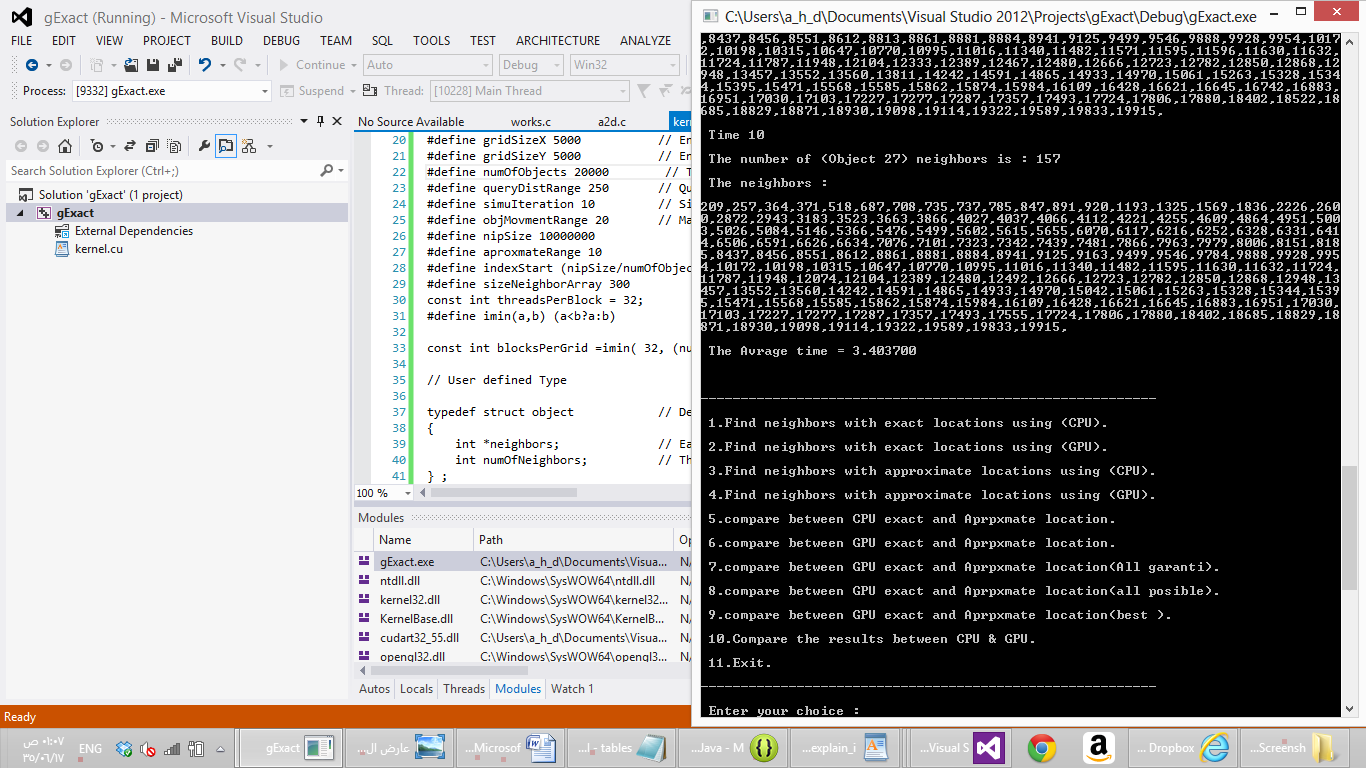
Number of objects = 5000 in CPU



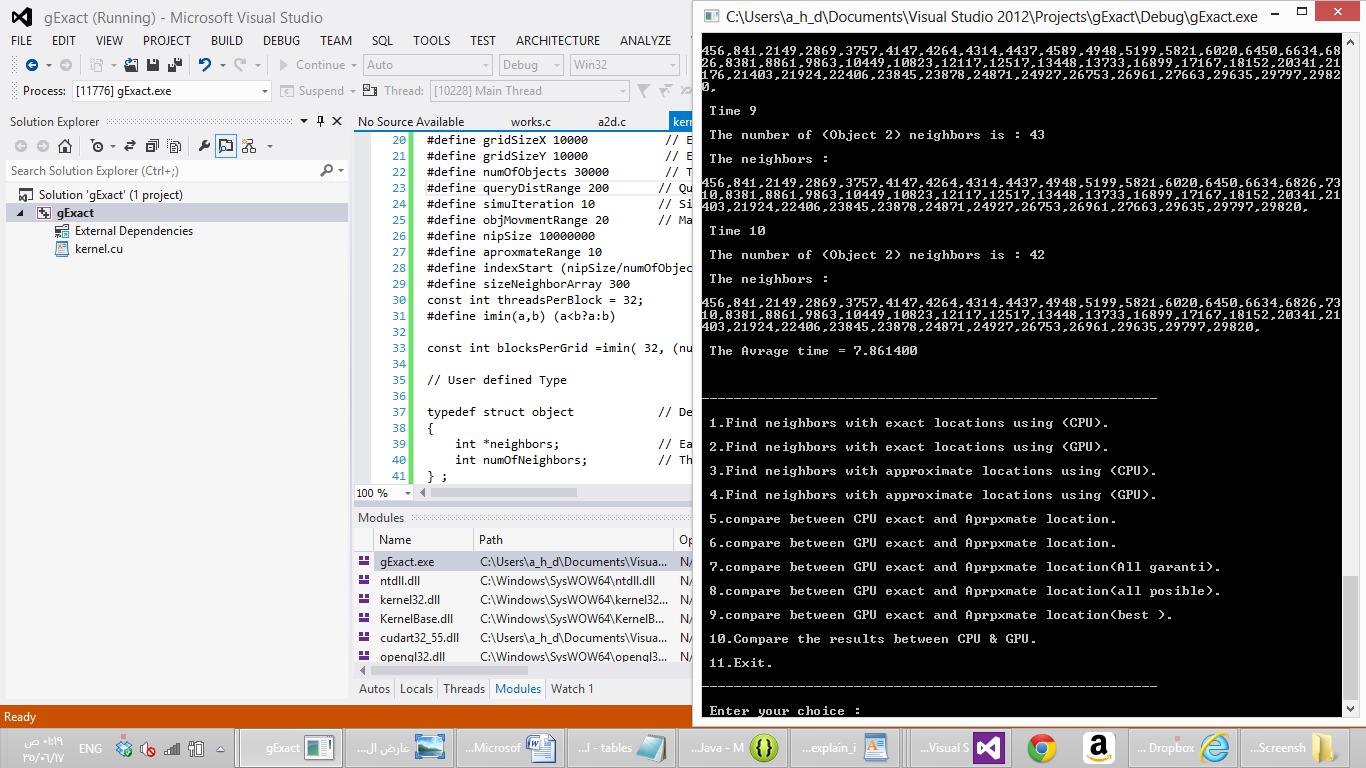
Number of objects = 10000 in CPU



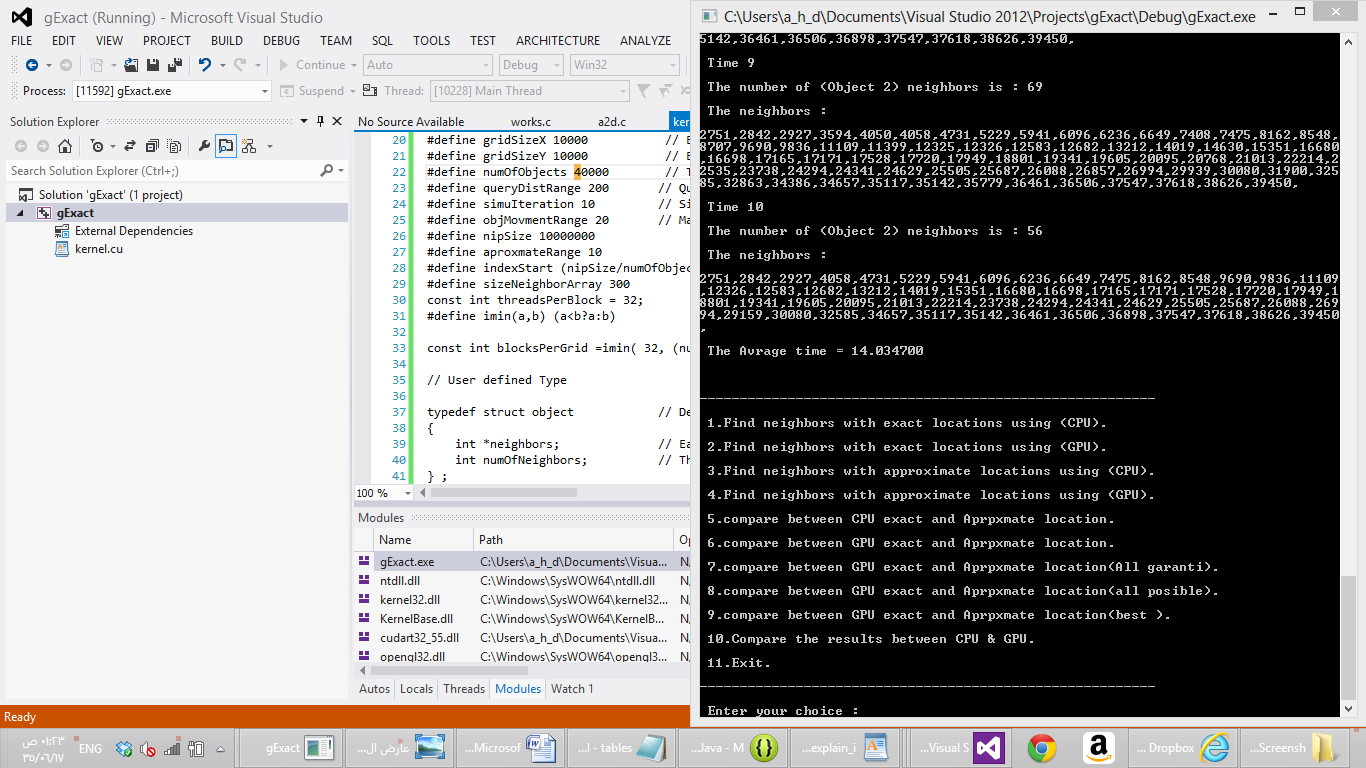
Number of objects = 20000 in CPU



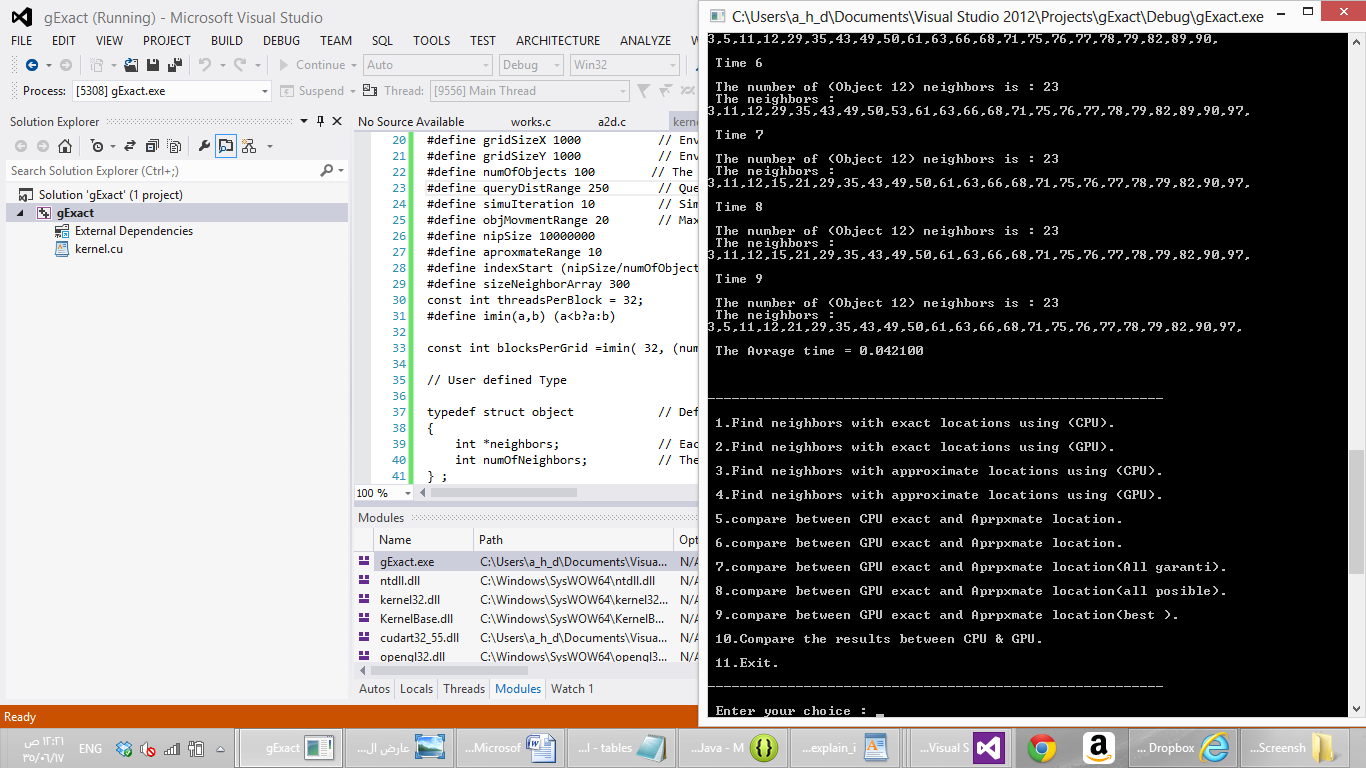
Number of objects = 30000 in CPU



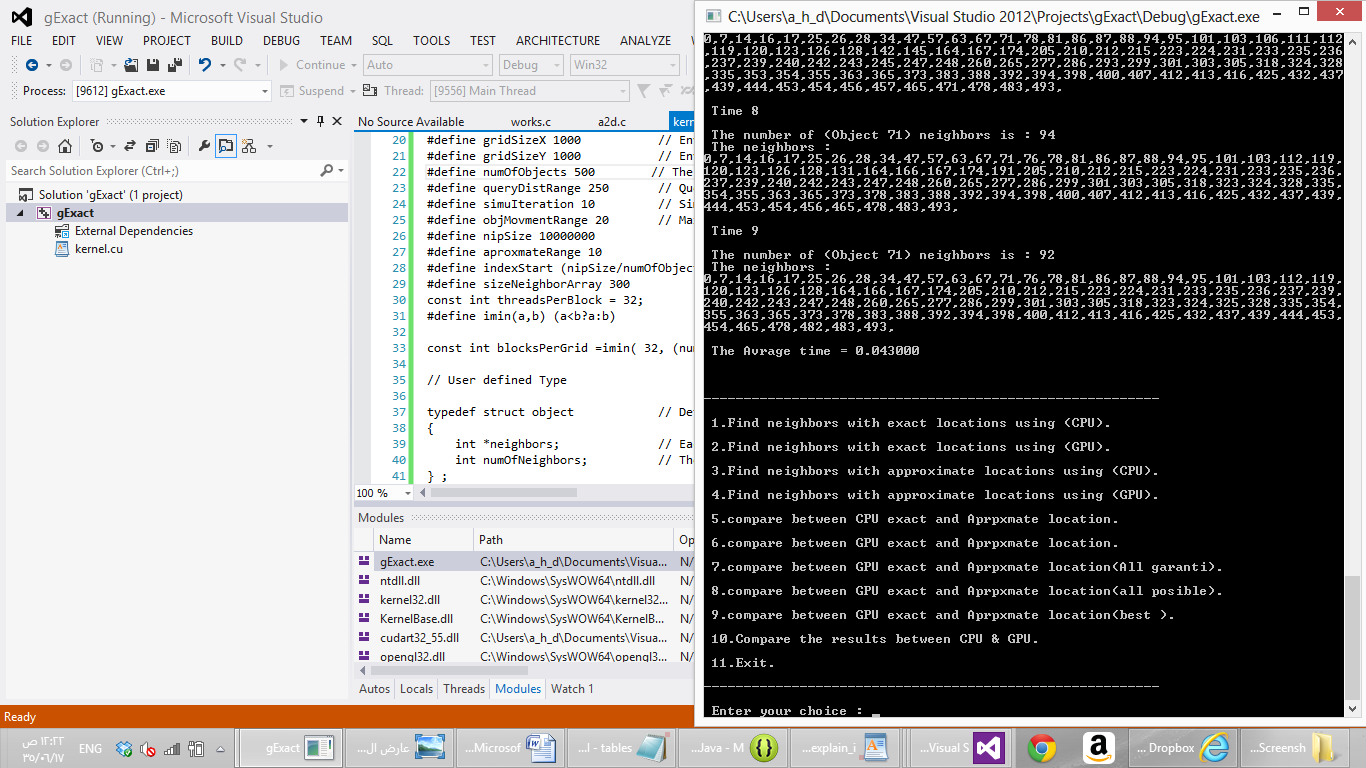
Number of objects = 40000 in CPU



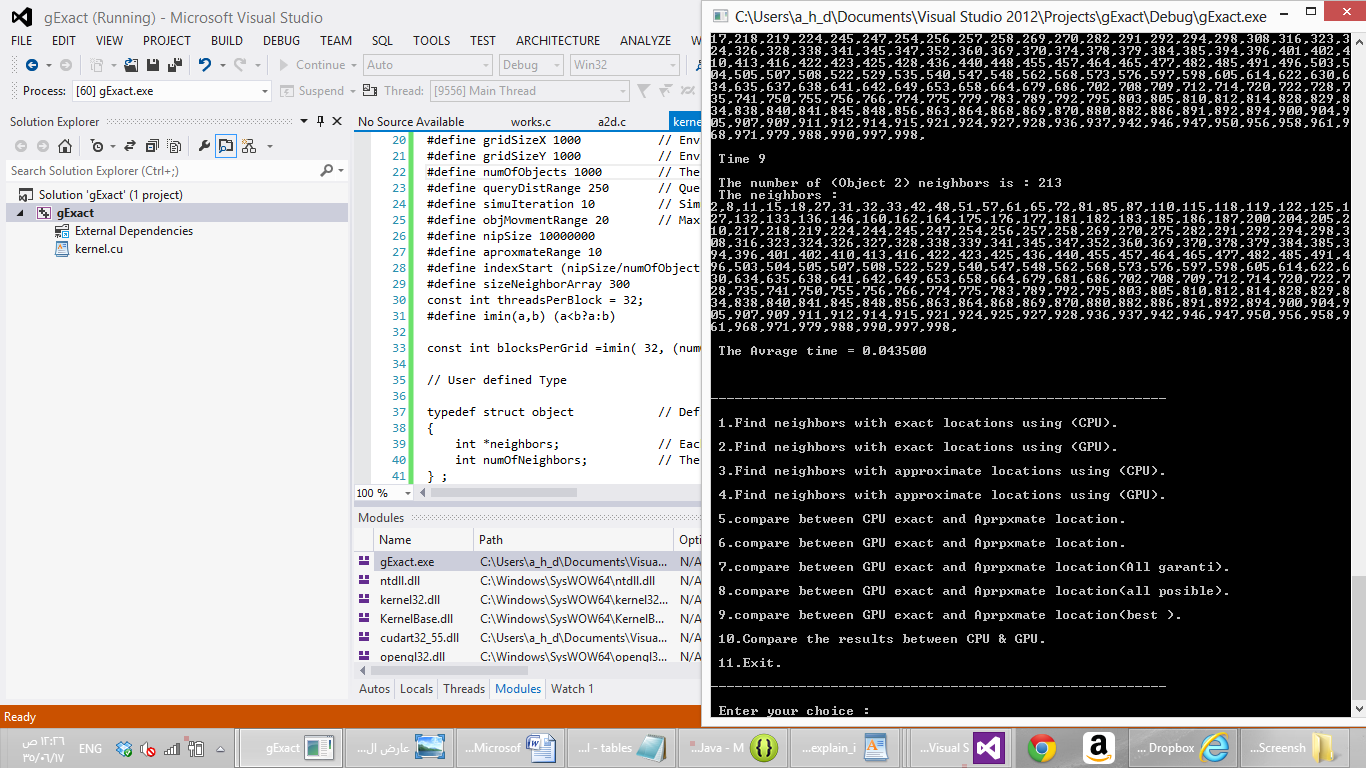
Number of objects = 100 in GPU



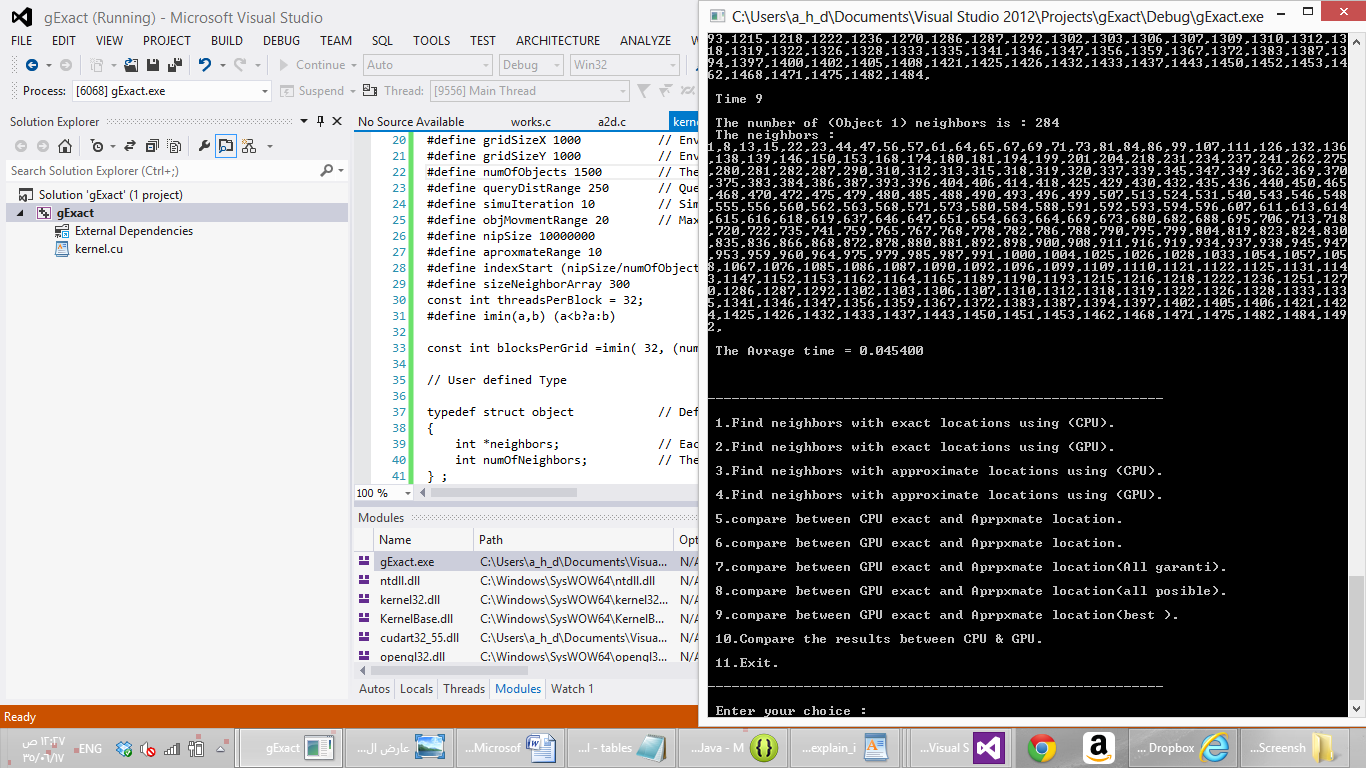
Number of objects = 500 in GPU



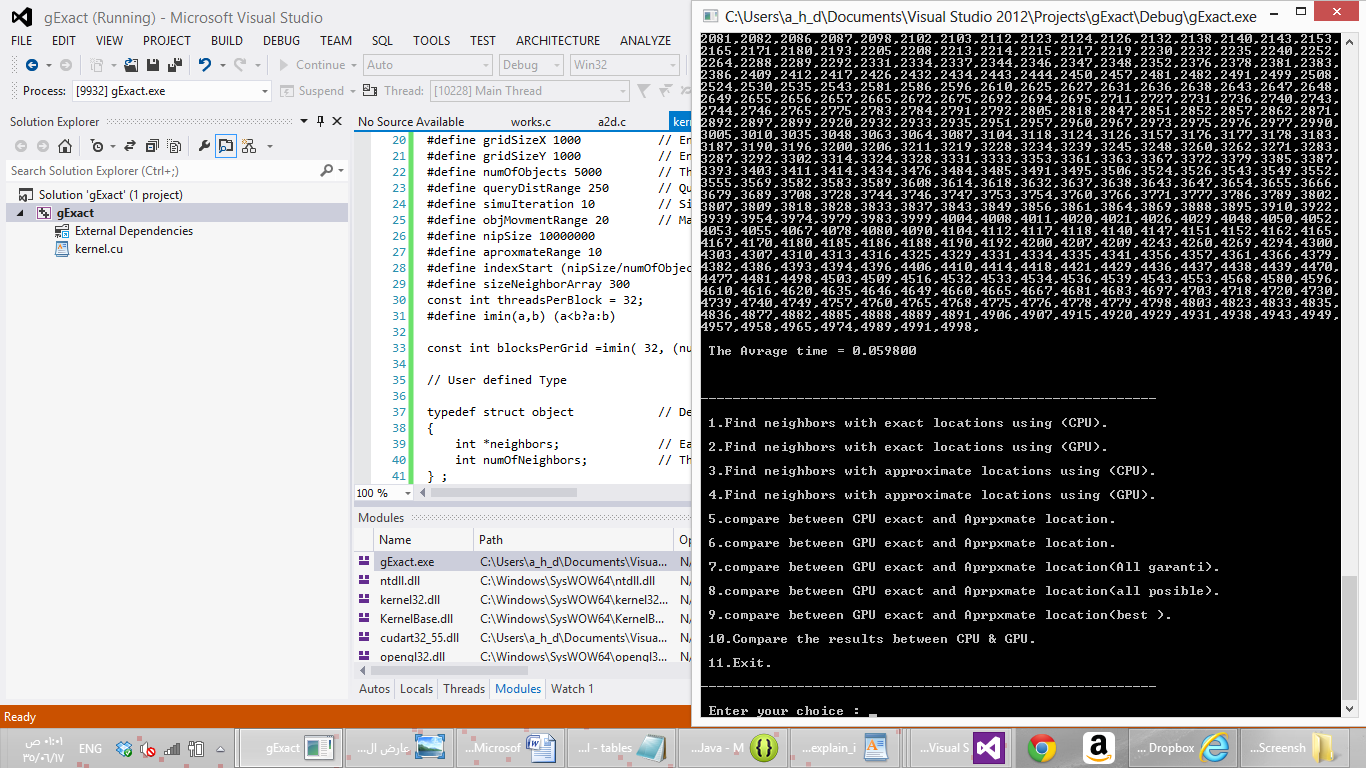
Number of objects = 1000 in GPU



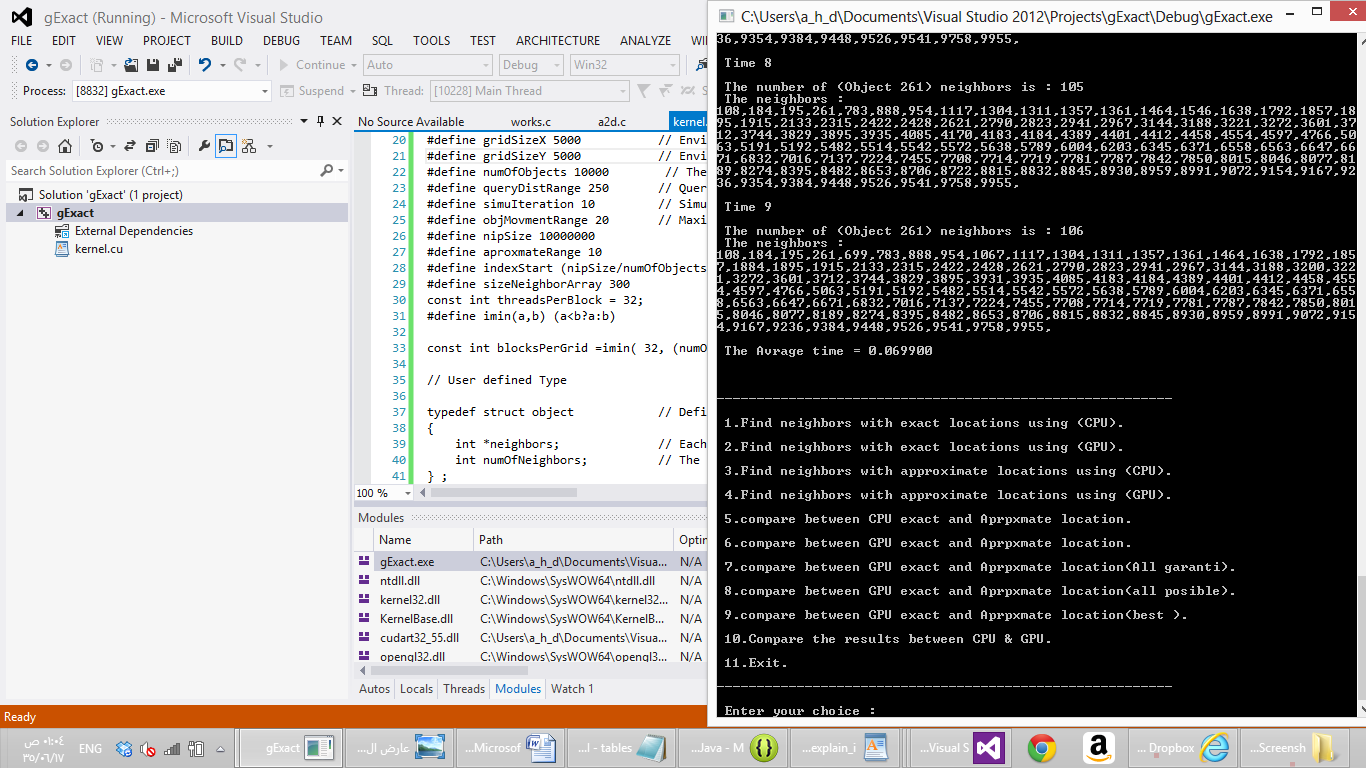
Number of objects = 1500 in GPU



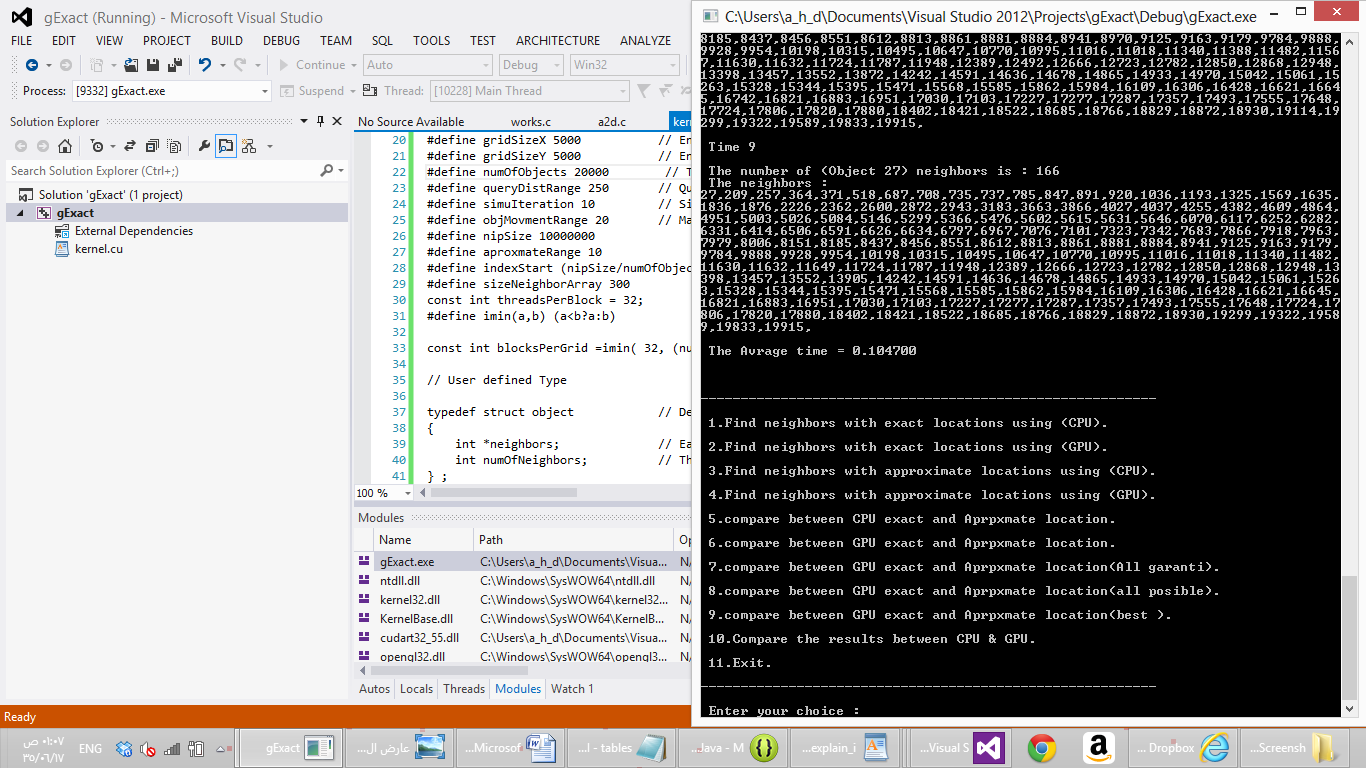
Number of objects = 5000 in GPU



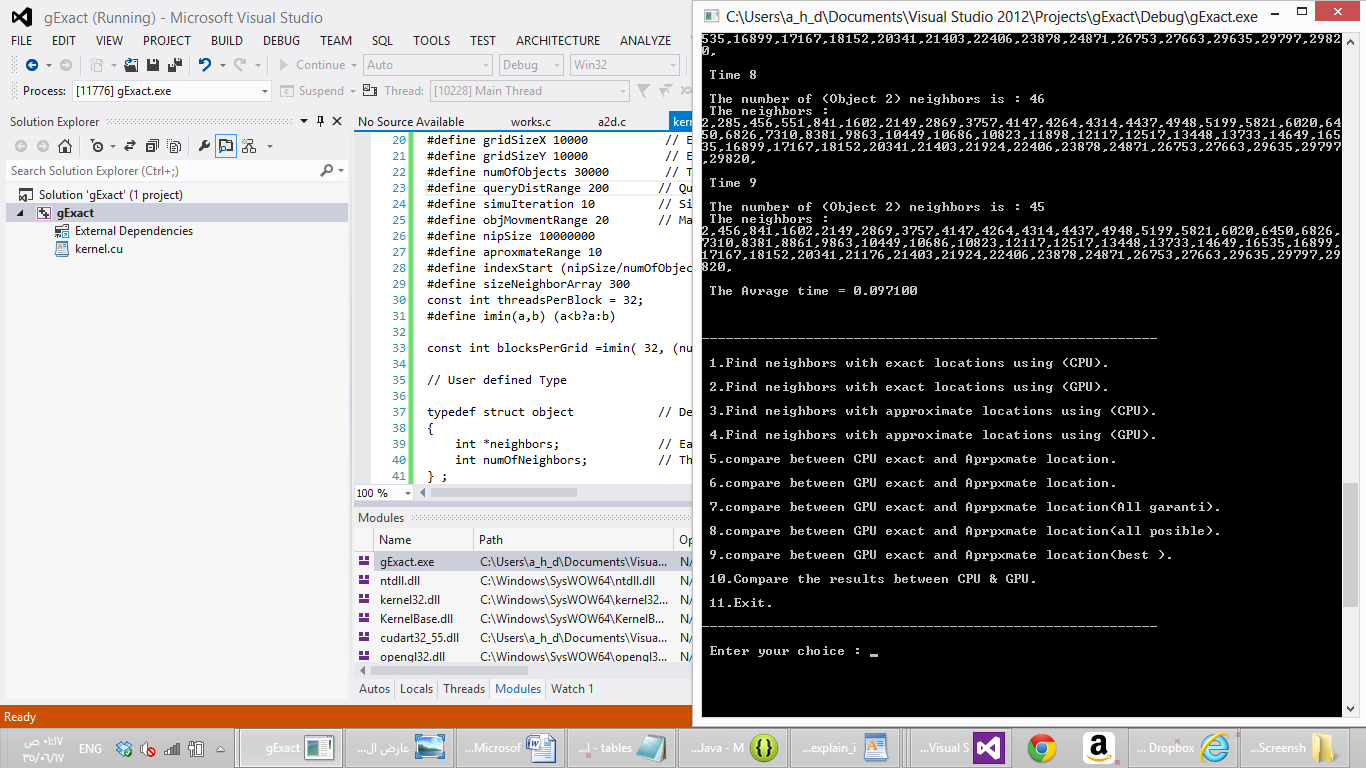
Number of objects = 10000 in GPU



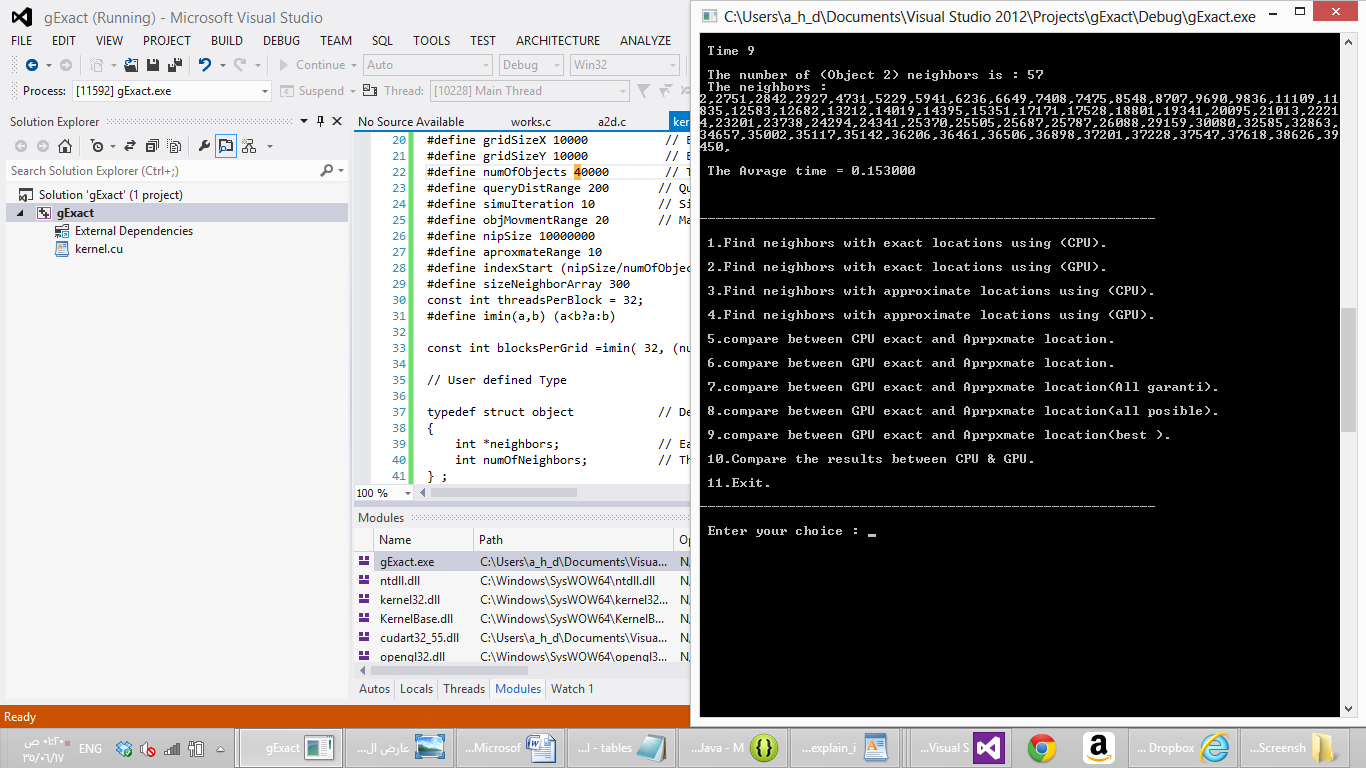
Number of objects = 20000 in GPU



Number of objects = 30000 in GPU



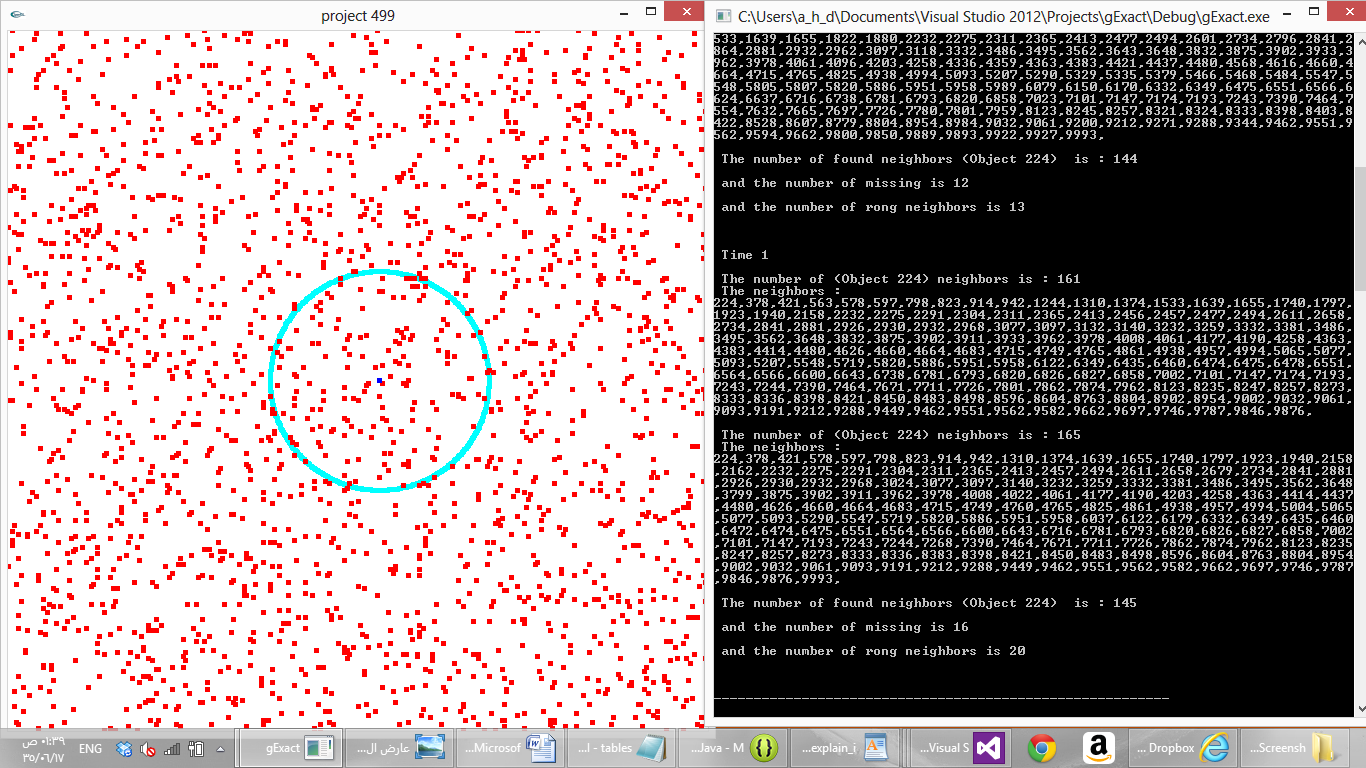
Number of objects = 40000 in GPU



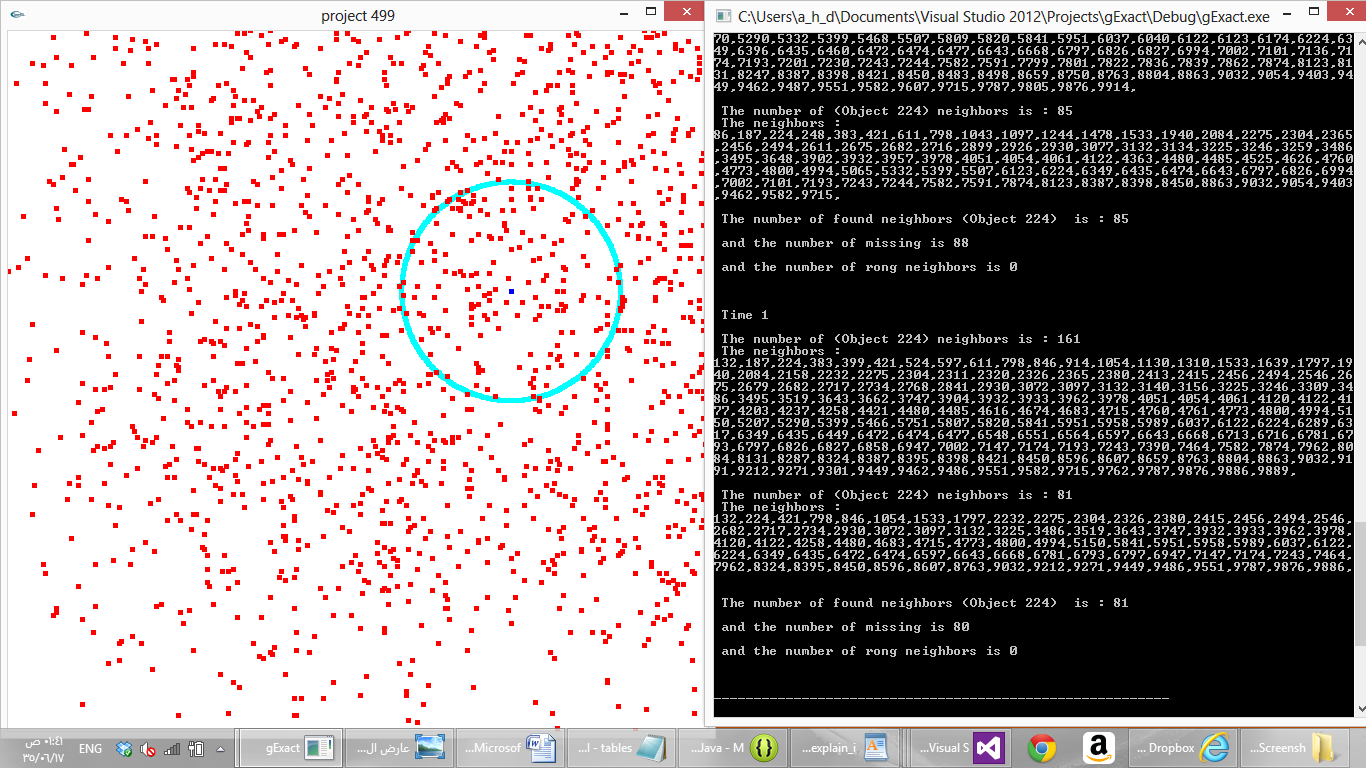
* Location : Approximate

Number of objects = 10000

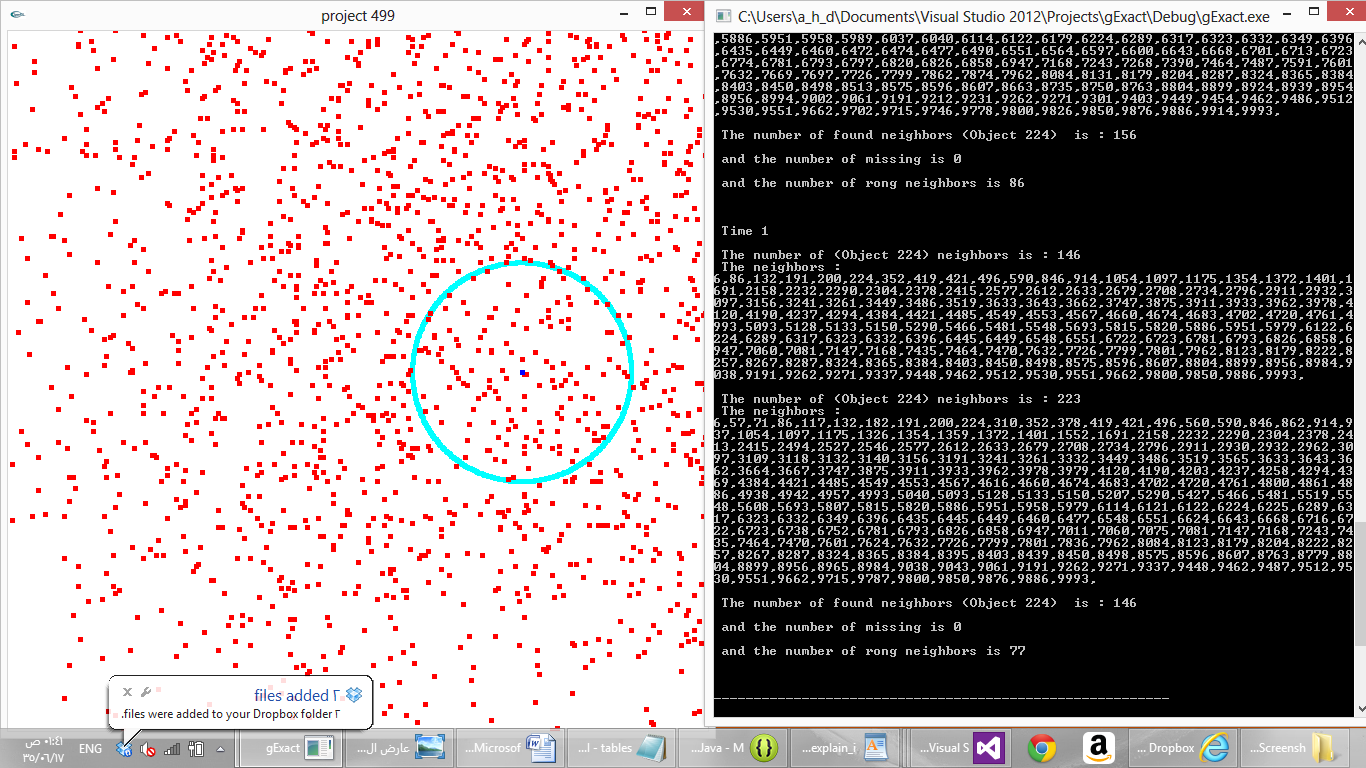
Compare the results between CPU exact and approximate locations.



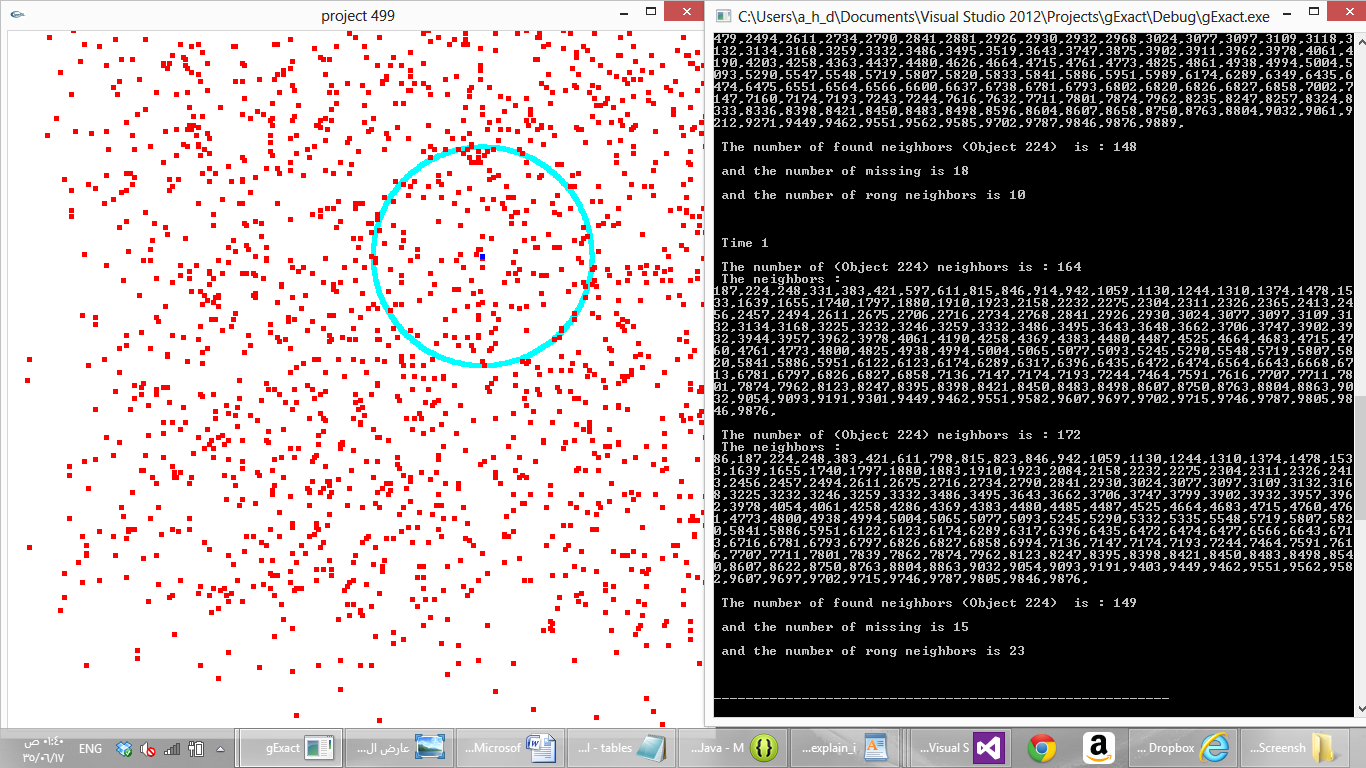
Compare the results between GPU exact and approximate locations (Guaranteed neighbors).



Compare the results between GPU exact and approximate locations (All possible neighbors).



Compare the results between GPU exact and approximate locations (Best approximation).



**Conclusion for testing phase**

The results were good ,reasonable and as expected, and we will try to improve it more so the system will work perfectly.

# References

**List of references :-**

## Research papers :

1. A. Aved, K. A. Hua, and A. Petkova, “Processing Approximate Moving Range Queries in Mobile Sensor Environments” , pp. 1-17 , CSE-09 international conference, 2009 – IEEE .
2. A. Aved, K. A. Hua, and A. Petkova, “CAWA: Continuous Approximate Where-About Queries”, pp. 1-6 , - Scalable Information Systems, 2009 – Springer.

## Book :

1. “CUDA by Example: An Introduction to General-Purpose GPU Programming”, J. Sanders, E. Kandrot, NVIDIA Corp, 2010.

# Code ( Copy and Paste all of your project code…..)

// Definning identifires

#define gridSizeX 1000 // Environment x-axis size

#define gridSizeY 1000 // Environment y-axis size

#define numOfObjects 10000 // The number of objects at the environmet

#define queryDistRange 100 // Query range in x-axis & y-axis

#define simuIteration 10 // Simulation iterations (number of simulation iterations)

#define objMovmentRange 30 // Maximum movment range in the x-axis & y-axis

#define nipSize 10000000

#define aproxmateRange 10

#define indexStart (nipSize/numOfObjects)

#define sizeNeighborArray 300

const int threadsPerBlock = 32;

#define imin(a,b) (a<b?a:b)

const int blocksPerGrid =imin( 32, (numOfObjects+threadsPerBlock-1) / threadsPerBlock );

// User defined Type

typedef struct object // Define a type called object

{

int \*neighbors; // Each neighbor ID

int numOfNeighbors; // The number of neighbors for each object

} ;