# Lab 3 – OpenACC

Name 1: Alejandro González _____NIA: 252658

Name 2: Luca Franceschi _____NIA: 253885

## Delaunay Triangulation:

1. **Plot the execution time for different values of points (100, 1000, and 10000) and different image sizes (200x150, 800x800, and 2000x1500) for both the CPU and the GPU (using OpenACC) executions.**

This first table shows the execution time with different points and grid sizes for the sequential version:

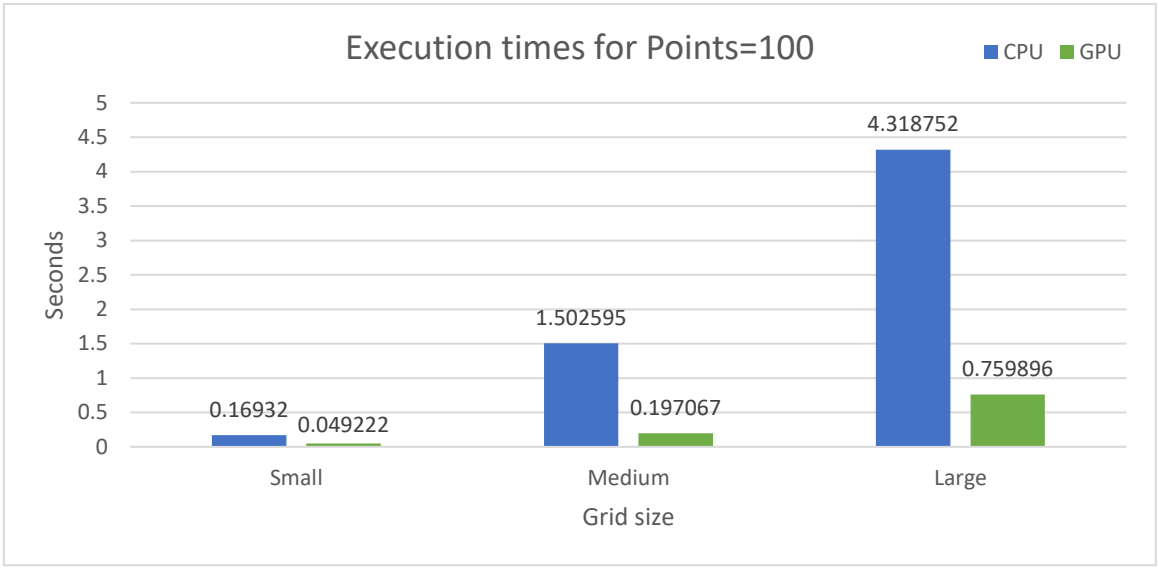
| CPU time (seconds) | 100p | 1000p | 10000p |
|---|---|---|---|
| 200x150 (Small) | 0.16932 | 12.851149 | 11233.10214 |
| 800x800 (Medium) | 1.502595 | 20.29842 | 10912.89852 |
| 2000x1500 (Large) | 4.318752 | 45.460451 | 11249.2815 |

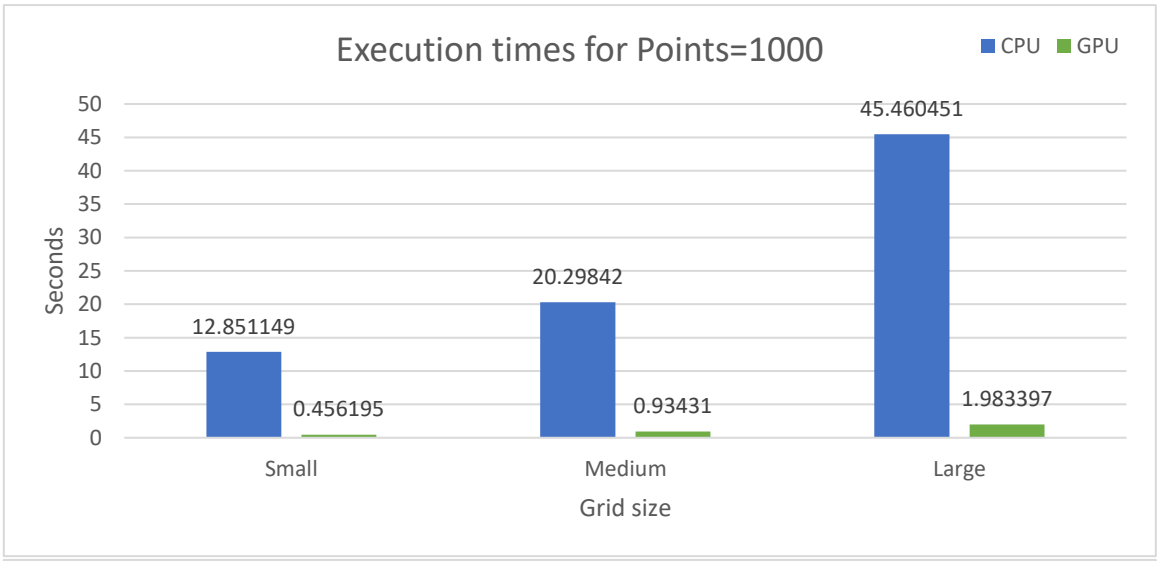
And this second table shows the execution time with the same points and grid sizes as before for the parallelized version:

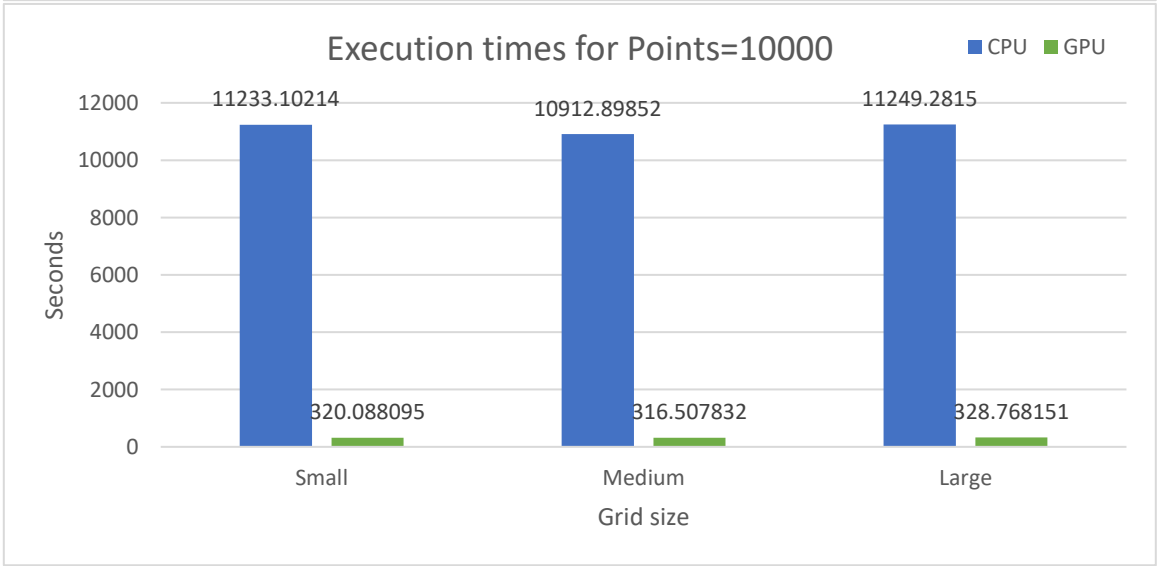
| GPU time (seconds) | 100p | 1000p | 10000p |
|---|---|---|---|
| 200x150 (Small) | 0.049222 | 0.456195 | 320.088095 |
| 800x800 (Medium) | 0.197067 | 0.93431 | 316.507832 |
| 2000x1500 (Large) | 0.759896 | 1.983397 | 328.768151 |

At first the only thing that we can clearly see is that the speedup is very significant for the largest versions of the problem: from around 3 hours for the longest sequential problem to around 5 minutes. It is important to note that this tables represent the total execution time of the program (count close points + Delaunay triangulation + generate image).

When we plot this data, we get something like the following:


Execution times for Points=100
CPU
GPU
Seconds
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0.16932
0.049222
1.502595
0.197067
4.318752
0.759896
Small
Medium
Large
Grid size


Execution times for Points=1000
CPU
GPU
Seconds
50
45
40
35
30
25
20
15
10
5
0
12.851149
0.456195
20.29842
0.93431
45.460451
1.983397
Small
Medium
Large
Grid size


Execution times for Points=10000
CPU
GPU
Seconds
12000
10000
8000
6000
4000
2000
0
11233.10214
320.088095
10912.89852
316.507832
11249.2815
328.768151
Small
Medium
Large
Grid size

2. **Explain them.**

We can see from the plots above that the execution time on GPU is by far lower than the execution time on CPU. This makes sense as the CPU is more focused on making a single process really fast, and the GPU is more focused on making lots of processes work in parallel. However, the most noticeable thing here is how the relationship between the times of CPU vs GPU evolves when changing the number of points. To do this, we will analyze each case separately:

- 100 points.
    - 200x150 (small). The time for CPU is 0.16932 and time for GPU is 0.049222. Therefore, the **GPU is around x3.44 times faster**.
    - 800x800 (medium). The time for CPU is 1.502595 and time for GPU is 0.197067. Therefore, the **GPU is around x7.62 times faster**.
    - 2000x1500(large). The time for CPU is 4.318752 and time for GPU is 0.759896. Therefore, the **GPU is around x5.68 times faster**.
- 1000 points.
    - 200x150 (small). The time for CPU is 12.851149 and time for GPU is 0.456195. Therefore, the **GPU is around x28.17 times faster**.
    - 800x800 (medium). The time for CPU is 20.29842 and time for GPU is 0.93431. Therefore, the **GPU is around x21.73 times faster**.
    - 2000x1500(large). The time for CPU is 45.460451 and time for GPU is 1.983397. Therefore, the **GPU is around x22.92 times faster**.
- 10000 points.
    - 200x150 (small). The time for CPU is 11233.10214 and time for GPU is 320.088095. Therefore, the **GPU is around x35.09 times faster**.
    - 800x800 (medium). The time for CPU is 10912.89852 and time for GPU is 316.507832. Therefore, the **GPU is around x34.48 times faster**.
    - 2000x1500(large). The time for CPU is 11249.2815 and time for GPU is 328.768151. Therefore, the **GPU is around x34.22 times faster**.

By looking at these numbers, we can clearly see that when we increase the points (and thus we increase the number of calculations that have to be done because of the nested for loops), the increase in efficiency gets much more relevant. For example, in the case of 100 points, the GPU goes around 8 times faster in the best-case scenario, while in the case of 10000 points the GPU goes around 35 times faster in the best-case scenario. This is due to the fact that the GPU achieves its maximum performance when it has enough workload to exploit the parallelism that offers its architecture. When we put 100 points in the GPU, we don't exploit its capabilities as well because the time to load to GPU and back to CPU when done is not worth it for the little calculations we are going to perform on GPU. We have to remember that the exchange of data between the CPU and GPU is done by a bus and thus it can be a bottleneck for performance in certain cases.