Parallel Programming

Sergio Oyaga & Carlos Mougan

GPU

Submitted To:
Parallel Programming
Computer Science
Department

Submitted By :
Sergio Oyaga
Carlos Mougan
MSc in Modelling
CRM

1 Motivation & Description

The goal of this assignment is to understand the main differences of CPU and GPU that are two quite different processing units, designed for two different purposes, and they have different performance characteristics. Certain tasks are faster in a CPU while other tasks are faster computed in a GPU.

In a first simple approach, one can think that CPU is good for complex manipulations with small data & GPU is better when doing simple manipulations in a large data set.

This assignment is divided into two parts:

- Solving a linear system of equations using the Jacobi Method
- Heat Transfer

2 Solving a linear system of equations using the Jacobi Method

In this section, we will focus on understanding the trace and analysis files. Also, we will try to understand the result. We won't focus on the temporal analysis, that will be in the second part.

We will use Nvidia visual profiler in order to analyze the trace and analysis file. And this section is going to be the focus on the understanding of this tool.

In order to optimize the usage of the visual tool, we will try to execute a problem which sizes are enough to make the analysis file have the required amount of data. Unfortunately, this was not achieved that's why we use the given trace and analysis files given by the professors.

2.1 Our Attempt

We have used the following command from the terminal in order to automate a way to execute at the same time in the CPU & GPU all the needed files.

```
1   \$ sbatch -w aomaster -o res.txt bash.sub
```

Our size proposals where:

- Nsize = 1000 Max_Iterations=100

In Figure 1 we can appreciate the first output provided by the visual tool.

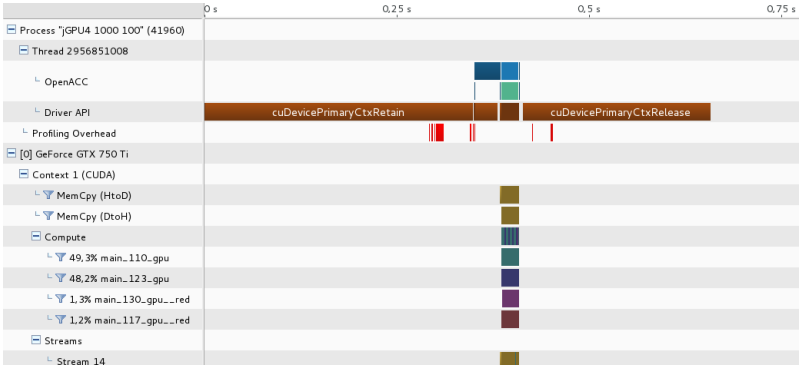


Figure 1: GPU Size 1000 & Iter 100

The first thing that notice is the short time in the execution time in comparison of the longer time to prepare the device, which in this case is GeForce GTX 750 Ti.

In order to appreciate the way the kernels work we do a zoom in the time scale.

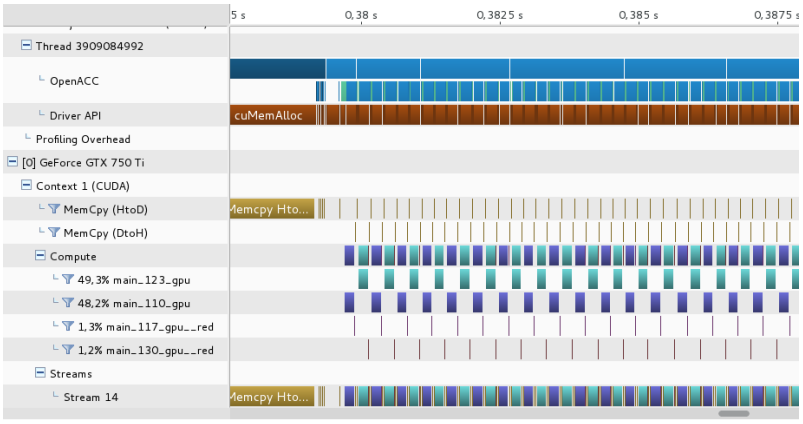


Figure 2: Zoom GPU Size 1000 & Iter 100

In figure 2 we can see a fraction of the total time where the kernels are working and we can appreciate which kernels work in parallel with whom.

Although we wanted to analyze specifically the different kernels (memory usage, bandwidth...), the execution times of each process in each kernel are too short. That's why we introduce now the solution given by the professor.

2.2 Using the Given Example

In the virtual campus, we were able to get to two none optimal solutions of the Jacobi method. In this files, we were able to analyze the kernels independently.

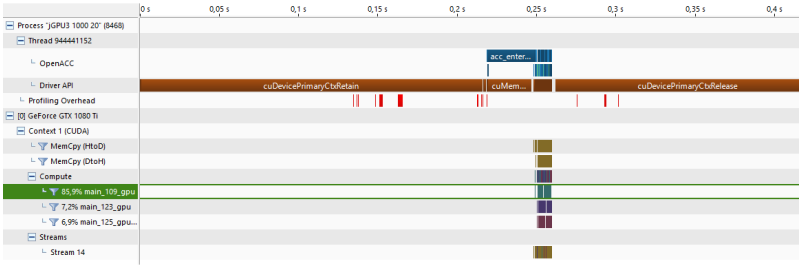


Figure 3: GPU 1000 20

In Figure 3 we can see the no totally optimized performance given for the teachers.

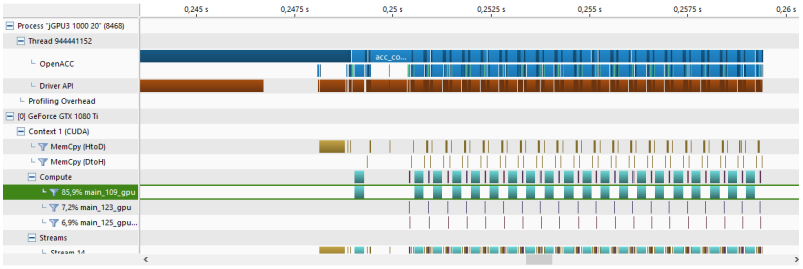


Figure 4: Zoom GPU 1000 20 and critical kernel in green

On the other hand, now we can visualize the kernel performance. We will try to focus on the kernel that takes longer (also known as the critical kernel).

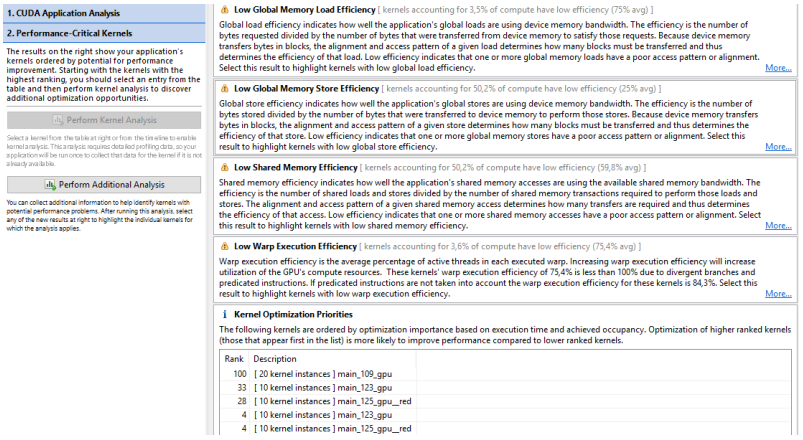


Figure 5: Kernel Analysis

Figure 5 is a screen capture of where the principal issues of the kernels are presented and fortunately, Nvidia tool gives us advice on how can we fix it.

We will like to focus on the critical Kernel since this is the one that determines the total execution time.

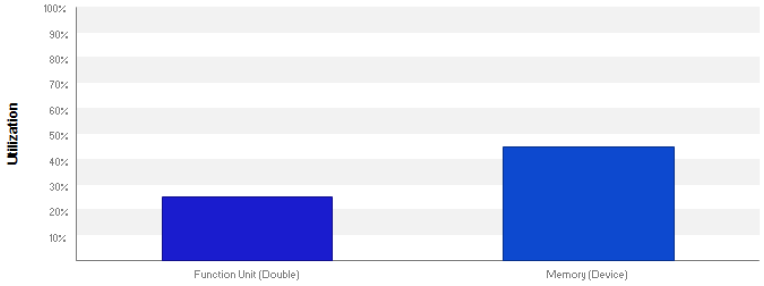


Figure 6: Kernel Performance

In Figure 6 we can see that the utilization of the function unit is the half of the memory with this graphical representation we can deduce the low compute throughput and memory bandwidth utilization relative to the peak performance of the GeForce GTX 1080 Ti. Typically this indicates latency issues.

After seeing Figure 6 the tool recommends us to do a latency performance given that this is the principal problem (Figure 7).

Variable	Achieved	Theoretical	Device Limit	Grid Size: [1000,1,1] (1000 blocks)Block Size: [128,1,1] (128 threads)
Occupancy Per SM				
Active Blocks		16	32	
Active Warps	59,82	64	64	
Active Threads		2048	2048	
Occupancy	93,5%	100%	100%	
Warps				
Threads/Block		128	1024	
Warps/Block		4	32	
Block Limit		16	32	
Registers				
Registers/Thread		24	65536	
Registers/Block		3072	65536	
Block Limit		21	32	
Shared Memory				
Shared Memory/Block		1024	98304	
Block Limit		96	32	

Figure 7: Critical Kernel Performance

Although the characteristics of the problem allow the device to use all of the wraps the acquired level is not the maximum which it traduces as latency.

In figure 7 we can see the kernels block size, register usage, and shared memory usage. All this lets us know that the occupancy is not limiting the performance of the kernel.

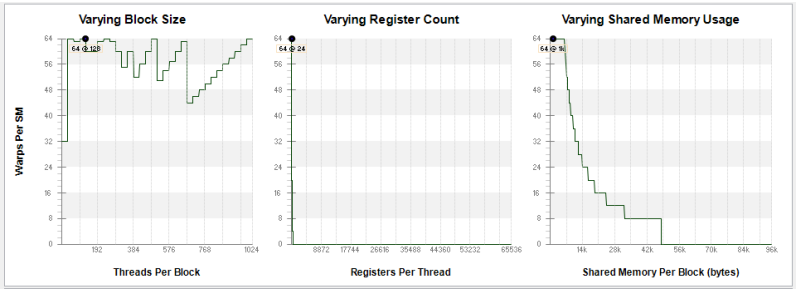
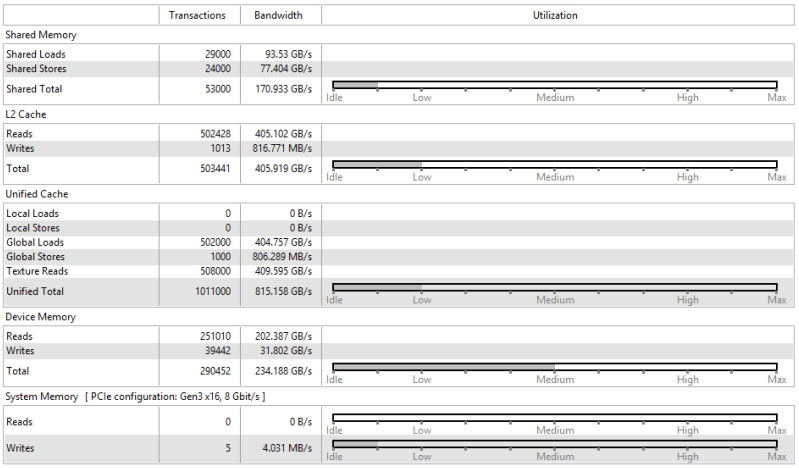


Figure 8: Performance Visualization of the critical kernel

In figure 8 we can see a graphical visualization of the variables just explained.

Another issue that can happen but doesn't occur is problems with the bandwidth size, uppon the charge that we give to the GPU device, the information transmission can overload it. In figure 9 we can appreciate this last analysis.



	Transactions	Bandwidth	Utilization
Shared Memory			
Shared Loads	29000	93.53 GB/s	
Shared Stores	24000	77.404 GB/s	
Shared Total	53000	170.933 GB/s	Idle Low Medium High Max
L2 Cache			
Reads	502428	405.102 GB/s	
Writes	1013	816.771 MB/s	
Total	503441	405.919 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	502000	404.757 GB/s	
Global Stores	1000	806.289 MB/s	
Texture Reads	508000	409.595 GB/s	
Unified Total	1011000	815.158 GB/s	Idle Low Medium High Max
Device Memory			
Reads	251010	202.387 GB/s	
Writes	39442	31.802 GB/s	
Total	290452	234.188 GB/s	Idle Low Medium High Max
System Memory [PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	4.031 MB/s	Idle Low Medium High Max

Figure 9: Memory Bandwith and utilization

Once seen utility of the Nvidia visualization tool we can forward to resolve the number 2 exercise.

3 Heat Transfer

Before starting with the analysis its necessary comment on the structure of the problem. The heat transfer problem in 1 dimension, its a spatial loop nested in a temporal loop. Our solution subdivides the spatial loop into tow different loops one for even temporal loops and the other one for odd temporal loops.

Trying to improve the memory accesses the first part of the analysis consists in analyzing again the kernels.

3.1 Kernel performance

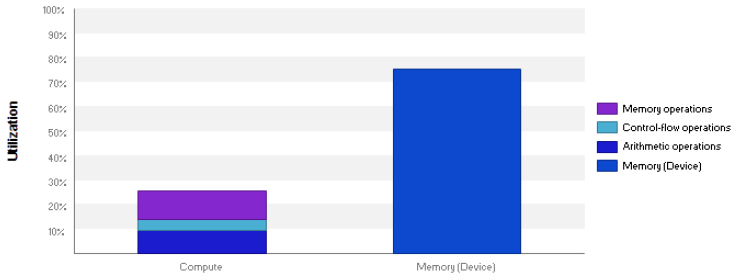


Figure 10: Critical Kernel performance

In contrast with the analysis of the last case here the main principal problem is not the latency it's the bandwidth saturation of the device.

Given this, we will analyze the bandwidth.

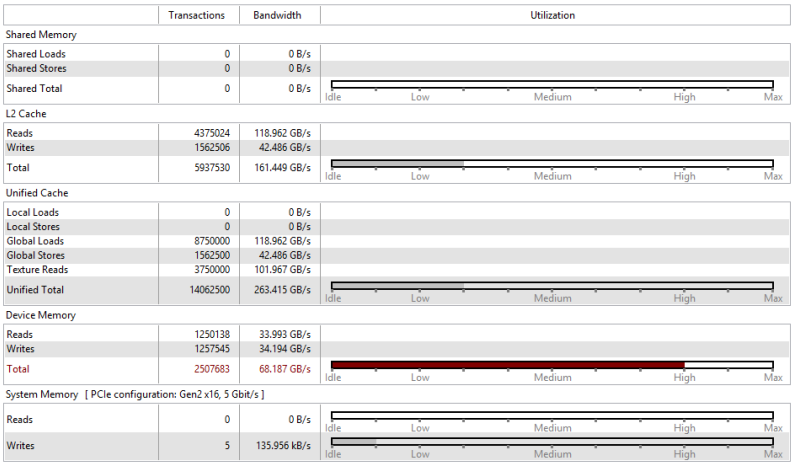


Figure 11: Memory Bandwith and utilization

In Figure 11 we can see the mentioned issue in red. For future improvement, we could try to improve cache memory and shared memory loads minimizing the device load.

3.2 Time Performance

In this section, we will analyze different optimized versions of the 1D heat transfer problem.

3.2.1 Basic Solution

The basic solution consists in applying the idea explained before of the nested loops. Logically we can analyze in a separated way the computational costs varying bots, spatial and temporal loops. As we can see in figure 12.

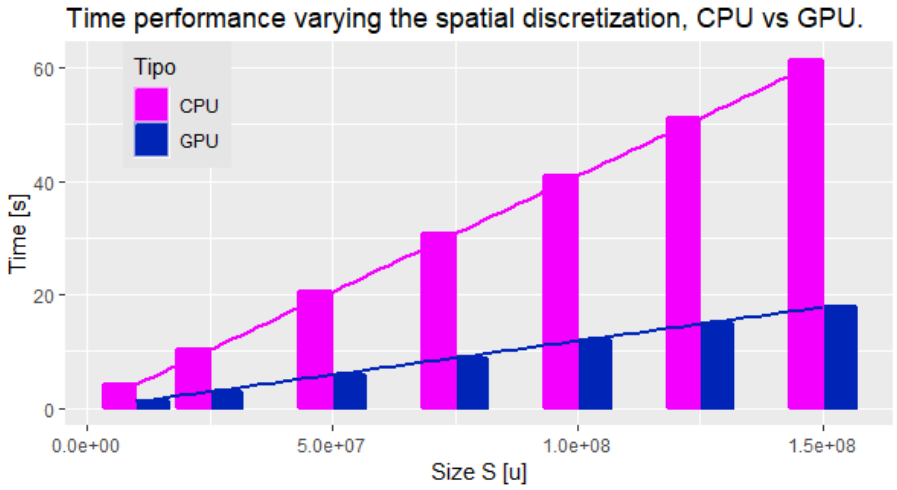


Figure 12: Time performance analysis changing the spatial loop

In this picture, we can see the time performance varying the spatial loop size for GPU and CPU. We can see how the CPU times are much greater than GPU ones. Also, both behave linearly.

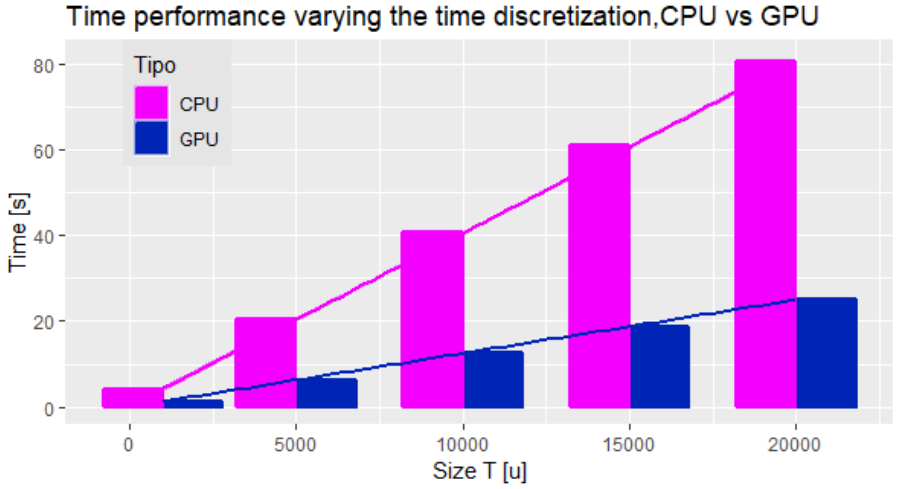


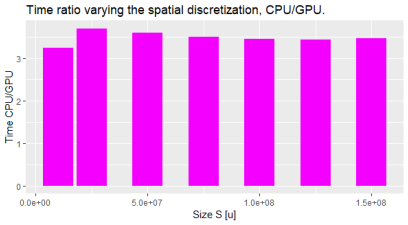
Figure 13: Time performance analysis changing the time loop

As before for the time loop variation, the CPU times are greater than the GPU ones and the behavior is also linear.

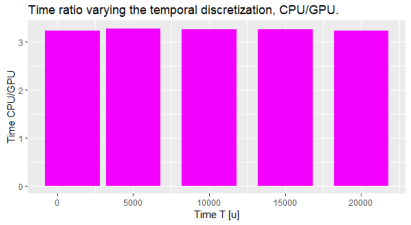
Even after this fact, the variation is stronger here than in the before the figure.

Even if we could think that the time ratio improves as we increase the loop sizes this is not true as we can see in the figure.

Time Ratios (CPU/GPU) for spatial and time loop variations



(a) GPU/CPU for spatial loop



(b) GPU/CPU for time loop

3.2.2 Time blocking optimization

The time blocking optimization consists in reducing the number of time steps into the half, third . . . clustering different temporal loops in the same. This means that for each time step instead of computing each time step the execution time will be reduced by the half.

Computing two time steps in one mean expressing the third time step in terms of the first avoiding the second one.

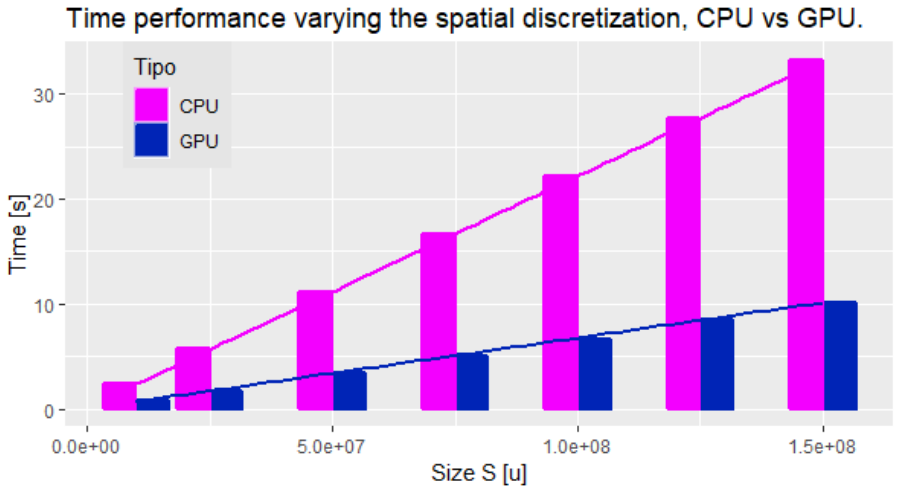


Figure 15: Time performance analysis changing the spatial loop

As we said the time has been reduced to the half for both executions CPU and GPU. Even if we can think that the temporal ratios can change as we see in the following picture, that's not accurate.

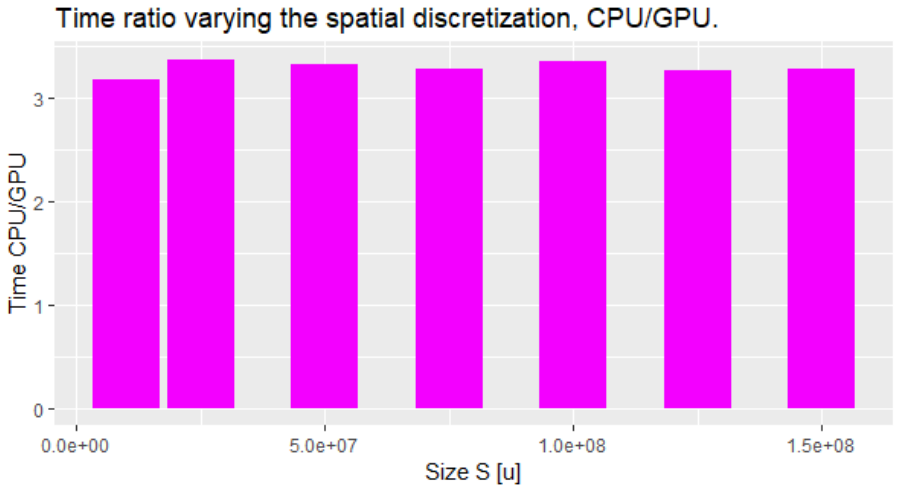


Figure 16: Time ratios GPU/CPU for spatial loop

3.2.3 Loop Simplification

As we understand the loops simplification consists in a reduction to the maximum level the number of operations realized in the inner loop.

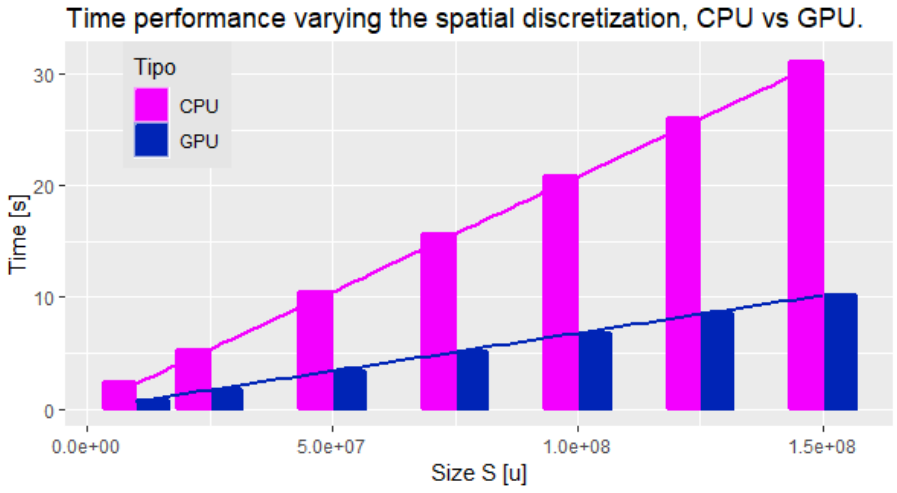


Figure 17: Time performance analysis changing the spatial loop

Again the ratios have not considerably changed.

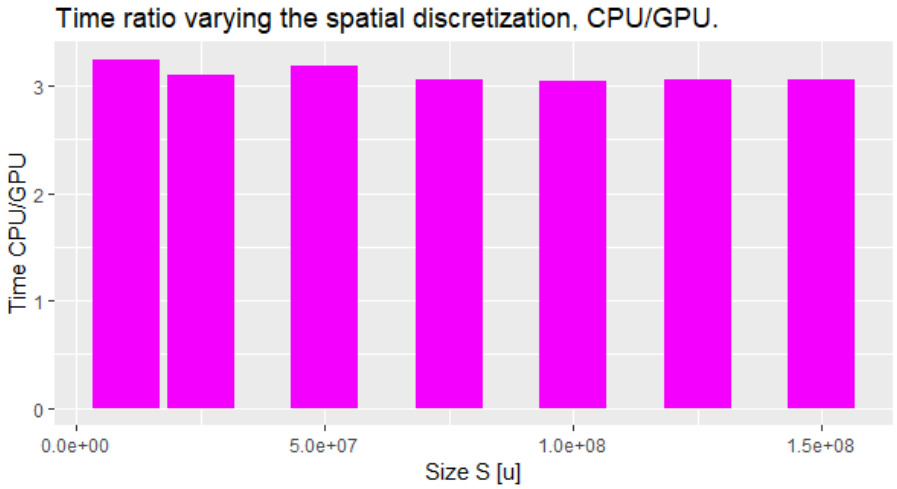


Figure 18: Time ratios GPU/CPU for spatial loop

It seems necessary to represent all the time ratios together to be able to compare if they are actually equal or not.

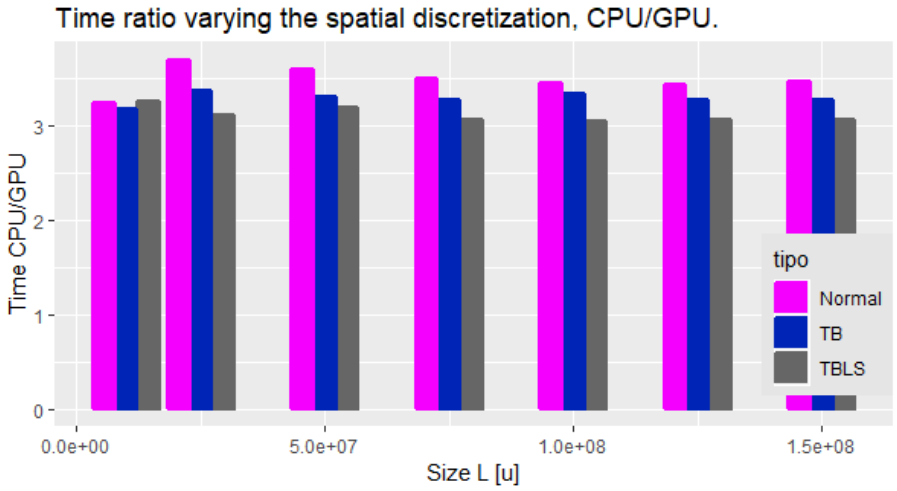


Figure 19: Time ratios comparative

We can see that systematically, although each improvement in the code achieves a better time, the CPU/GPU ratio slightly decreases.

4 Conclusions

In this project where we have tried to analyze the differences in the performance of the GPU and CPU with two different methods: Jacobi Method and Heat Transfer Equation.

In the first problem, we were able to gain an understanding of the Nvidia visualization tool, which is a powerful environment that enables us to see the GPU performance in depth. This tool not only analyzes the results but it also offers possible improvements that we apply to our parallelization.

In the second problem, we learned that when it comes to doing harsh computations the GPU offers the possibility to notorious accelerate our computation. Also in this section, we have done improvements in the code trying not to confront the device limitations. Even though we slightly increase the principal cause of out computational times, in its visualization.

To sum up, we consider GPUS a very powerful device to compute hardcore calculations, although its usage can be a little bit tricky at the beginning.