PACS LABORATORY 6

by

Daniel Ariza Antón (775545) and Marcos Ilarraza Sarto (797720)

The objective of this report is to explain the structure resulting from adapting the environment from the previous laboratory to make it work with two devices in order to flip a large amount of images.

Design decisions

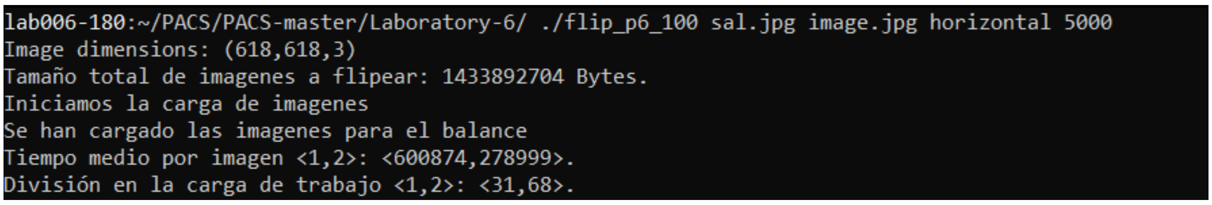
First, we had to deal with the problem of memory. We have been asked to apply our kernel to a large amount of images, 5000 to be exact. But it poses one question: How many images can we store in memory at the same time? First we tried to store all the 5000 images in dynamic memory, but the program failed due to segmentation fault. Therefore, we decided to take an approach of loading a certain amount of images, and once the kernel was applied to them, load another batch of images. At first we defined a fixed **buffer** for 100 images in memory, but afterwards we made it dependent on an input parameter, making it configurable before for each execution.

The problem of memory also affected our criteria regarding the kernel. At first, we implemented a kernel that expected a variable amount of images at the same time to flip them. But since we were running out of time and we thought that memory in the devices could be more limited (also keeping in mind the size of the memory buffers), we decided to keep the kernel from the previous laboratory in order to avoid both the memory and debugging problems that it could have.

Regarding the balance of the workload, since in this case we simulate that we have 5000 images by loading the same image many times, all the working blocks have exactly the same amount of workload. This is also a good reasoning for a real example since when applying filters, the inputs tend to have the same dimensionality in a specific batch. Because of this, we decided to use the same first five images to measure the kernel's time of execution in each device. Once we get the average time of execution in each device, we decided the number of positions in the **buffer** correspond to each other with the following equation:

$$numElementsBuffer_x = sizeOf(buffer) * time_Y / (time_Y + time_X)$$

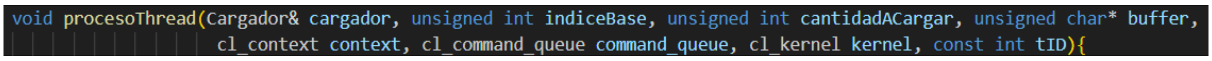
An example can be seen in the following terminal screenshot, were 1 is the CPU and 2 is the GPU:



```
lab006-180:~/PACS/PACS-master/Laboratory-6/ ./flip_p6_100 sal.jpg image.jpg horizontal 5000
Image dimensions: (618,618,3)
Tamaño total de imagenes a flipear: 1433892704 Bytes.
Iniciamos la carga de imagenes
Se han cargado las imagenes para el balance
Tiempo medio por imagen <1,2>: <600874,278999>.
División en la carga de trabajo <1,2>: <31,68>.
```

Therefore, a faster device will have more images charged in memory than a slower device. The times obtained for each individual test can be seen in the first table of the appendix (*Individual Kernel's time of execution for five images*).
In order to work with both devices, after balancing the workload as mentioned above, we decided to assign a thread to each of them so they can work in parallel. The balance of workload is done sequentially because we need the measures from both of them in order to use the equation mentioned before. Once the balance is done, they do not need to wait for the other kernel to finish, so the parallelization is the best option. There are still two things that they need to share though: the **amount of images left to load** and the **buffer** in which they can load them. The **buffer** (*buffer*) is passed by reference to each thread, alongside the range (*indiceBase* and *cantidadACargar*) that belongs to the device that is being used in it. Regarding the **amount of images left to load**, we decided to implement a class called *Cargador* which handles the loading of images to the buffer in the range specified. If the range can contain more images than the ones that are left, it just fills an amount equivalent to the number of images that were left and specifies it. The same instance of *Cargador* is shared between the threads.

The function that the threads exscute is called *procesoThread*, and its complete header is the following:



```
void procesoThread(Cargador& cargador, unsigned int indiceBase, unsigned int cantidadACargar, unsigned char* buffer,
                   cl_context context, cl_command_queue command_queue, cl_kernel kernel, const int tID){
```

This function defines the device and the kernel to be used with the parameters *context*, *command_queue* and *kernel*. The *tID* parameter is for printing on screen which thread is it. This function starts by loading in the **buffer** the images determined by the device's workload specified before. If no image has been loaded it finishes, otherwise it iterates along the **buffer** within its range to apply the filter to each of the loaded images and tries to load a new batch. In a normal implementation, we would save to disk the flipped images, but since that's not the objective of this laboratory we don't do so.

Once all the images have been flipped, we save one image for each device to check that both of them apply the filter correctly (specifically the first image, which should always have at least one image flipped from its last iteration).

Measures

In this section we will go through the different measures taken from the program and why we took them in what way. Unlike the previous lab, we have evaluated them with only one image size: 618x618 encoded in RGB. All the actual values can be seen in the tables in the Appendix. To understand them properly, we must keep in mind that the columns with the same **buffer size** correspond to the same experiment (Dev-1 buffer size 100 experiment 1 and Dev-2 buffersize 100 experiment 1 belong to the same call to the environment).

There are also complete terminal screenshots with examples of executions, were the threads IDs are 0 for the device 1 (CPU), and 1 for the device 2 (GPU).

Total time of execution per thread

The total time of execution ($\boldsymbol{T_{ex}}$) of each thread begins with the first instruction of the function *procesoThread* and ends with its last instruction. As in the previous lab, to measure the least amount of time we decided to measure the time with the ***clock command***, which only measures the time the program spent on the CPU and this doesn't include the time that the kernel spent working on its corresponding device. To measure the time it spent working we have used again the **cl_event** structure (adding the time of each call to the kernel). In this case, if we hadn't applied the workload balance, the **bottleneck** would be the CPU.

The medium time that each thread spent to flip all the images can be seen in the second table of the appendix (*total time of the thread*). As we can see, the difference between each pair of threads is 1000 milliseconds, which corresponds to one second. We consider that that difference in time is low enough, but it may be improvable.

Total number of flipped images per thread

As we can see in the fifth table (*number of images flipped per thread*), the amount of images flipped by each device is near the 50-50 division even though the workload that we assigned is bigger for the GPU than for the CPU. We believe that the reason is the cost of loading the images to the **buffer**. The workload that we have implemented assigns a number of images that each thread has loaded in memory at the same time. Since the workload is bigger for the GPU, its thread will spend more time loading images to the **buffer** than the CPU's. But thanks to being speeder than the CPU when applying the kernel, the total amount of time spent by each thread is almost equal. If we include the fact that the workload is a percentage value of the **buffer's size**, that is the reason why when using a bigger **buffer** (150 images) the medium number of flipped images with the GPU is smaller and the medium number of flipped images with the CPU is bigger. Because of this, we consider that the **bottleneck** is the load of images to the buffer. If we could have all the 5000 images in memory, the GPU would flip more images and as a result the program would end sooner.

Regarding the time that they would take independently, it would be equivalent to the previous lab measurements, but with these devices. So we made just an analytical calculation: By using a **buffer with 100 images** capacity, the total time to load the 5000 images would be equivalent for both the CPU and the GPU versions. Therefore, the main difference will be regarding the kernel's time of execution, which would be of *2,81775E+14ns* in the case of the CPU and *2,79044E+14ns* in the case of the GPU. The formula used for this calculation is the following:

$$Time_{Y,5000} = 5000 * meanTime_{Y}/meanFlips_{Y}$$

Where Y corresponds to the CPU or the GPU (Device 1 or Device 2) in the fourth (Kernel time of the thread) and fifth (*number of images flipped per thread*) tables.

Bandwidths

As for the bandwidths, we did the same that we did with the previous lab: measure the total time in each direction and divide the total number of bytes transferred by this value. As we can see in the sixth table(*Host->Device bandwidth),* the GPU transfers a mean of around 8 Bytes per nanosecond from the *Host memory* to *its memory*, while the CPU transfers a mean of around 10 Bytes per nanosecond. Therefore, in this case the **bottleneck** would be the GPU. Note that we are using the values from the test with a **buffer with capacity for 150 images**, because the amount of images flipped by each device is more similar. This is the value that we'll be using for the rest of the bandwidths.

Regarding the transferences from the *Device memory* to the *Host memory*, seventh table (*Device->Host bandwidth*), we can see that the GPU transfers a mean of around 14 Bytes per nanosecond, while the CPU transfers a mean of around 13 Bytes per nanosecond. By checking the values with a **buffer with capacity for 100 images**, we can clearly see that the GPU has a better performance with more images, and the CPU with less. Therefore, in this case the **bottleneck** would be the CPU.

Lastly, regarding the accesses to the global memory within the kernel's execution, eighth table (*kernel bandwidth*), we can check that the amount of Bytes transferred by nanosecond is pretty similar in both cases, with a difference of around 1E-7 Bytes. Because of this we consider that **none of them** would suppose a **bottleneck**, since their transference rate is pretty similar. This value is much slower than the one we gathered in the previous laboratory session. We think that the cause can be one of the following reasons:

- The device used was the *FPGA Emulation Device*, which could have a higher inner transference rate.
- Because it is calculated with the total number of images and the total kernel's time of execution.

We consider that is more probable the first reason, but we can't discard the second.

Adaptability to new execution cases

Since the **buffer** is resizable in each execution, the ideal size could be changed depending on the machine that it is being executed at. A downside is that the *procesoThread* function has fixed inputs regarding *flip_image* kernel. Therefore, in order to use the same function the kernel function should use the exact same parameters (and have the same name for the function and file as the *flip_image* kernel). Because of that, in order to use a different kernel it would be necessary to reimplement this functions (we think that there should be a way to make it independent of the kernel, and therefore avoid the need to compile the code again).

Regarding the kernel we used, we believe that the performance could be improved if we had finished the implementation of the kernel that can flip more than one image per call. The reason is that the number of calls to *clEnqueueWriteBuffer* and *clEnqueueReadBuffer* would be less, avoiding the costs of calling each of them more times with a smaller number of bytes each.

Lastly, our balance of workload works fine as long as the chunks of work are always the same (or at least most of them). Otherwise, the outcome is unknown, but most likely will be worse than with the mentioned case.

Appendix

Individual Kernel's time of execution for five images

	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	600874	589518	278999	279849
2	660197	652811	280033	281399
3	597292	592434	280599	279216
4	677144	692342	279733	279633
5	598834	652804	280533	280716
6	629615	599778	278666	278116
media	627326	629947,8333	279760,5	279821,5

Total time of the thread

	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
total time (ms)	140257407,3	140218582,8	140256121,2	140217464,2

Clock time of the thread

thread Time (ms)	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	54846	54602	54869	54625
2	54740	56292	54766	56326
3	56056	56166	56066	56132
4	54773	55888	54783	55905
5	55998	54911	55952	54838
6	56272	55957	56275	55971
media	55447,5	55636	55451,83333	55632,83333

Kernel time of the thread

kernel time (ns)	Dev-1 Kernel time 100	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	1,40059E+14	1,40559E+14	1,40058E+14	1,40558E+14
2	1,40533E+14	1,39651E+14	1,40532E+14	1,3965E+14
3	1,4068E+14	1,40681E+14	1,40679E+14	1,40679E+14
4	1,4038E+14	1,39738E+14	1,40378E+14	1,39737E+14
5	1,39921E+14	1,40238E+14	1,3992E+14	1,40237E+14
6	1,3964E+14	1,40111E+14	1,39638E+14	1,4011E+14
media	1,40202E+14	1,40163E+14	1,40201E+14	1,40162E+14

Number of images flipped per thread

imgs flipped	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	2485	2500	2515	2500
2	2487	2511	2513	2489
3	2491	2493	2509	2507
4	2486	2509	2514	2491
5	2491	2501	2509	2499
6	2487	2500	2513	2500
media	2487,833333	2502,333333	2512,166667	2497,666667

Host->Device bandwidth

Host->Dev (Bytes/ns)	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	9,48381	10,2584	8,47055	8,20775
2	9,98684	9,42078	8,79601	9,14678
3	10,2804	9,33429	8,03109	8,97393
4	9,24767	11,968	8,20852	8,50022
5	9,66732	9,89419	8,1767	9,58002
6	9,07217	9,69486	8,11498	8,15367
media	9,623035	10,09508667	8,299641667	8,760395

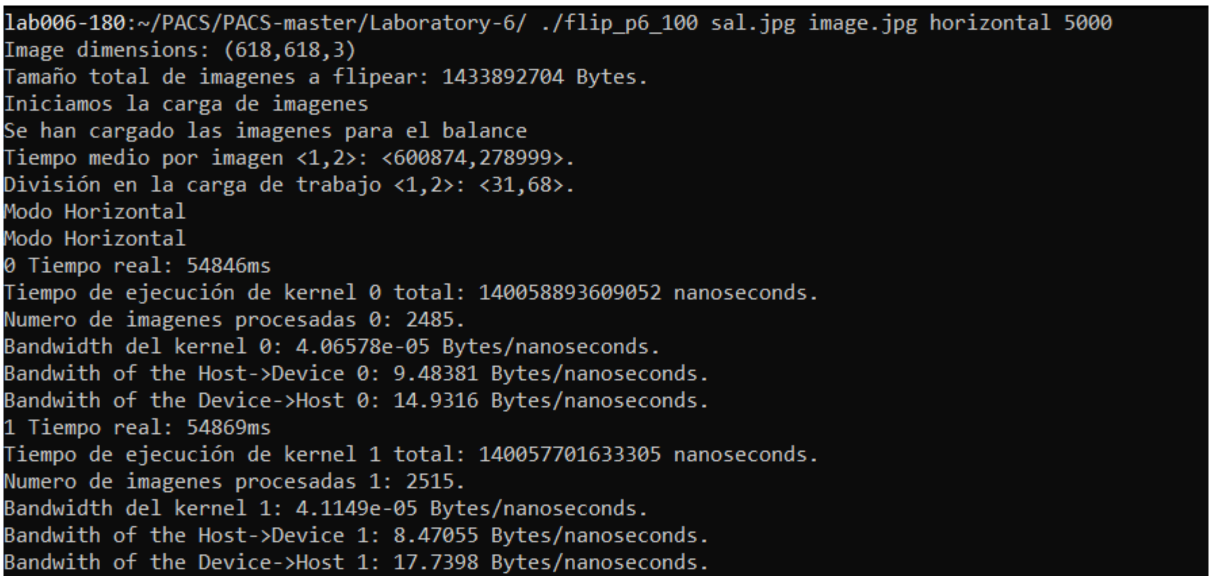
Device->Host bandwidth

Dev->Host (Bytes/ns)	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	14,9316	15,0655	17,7398	15,1833
2	13,539	11,892	17,0268	13,9032
3	14,562	13,92	17,459	15,1283
4	14,6373	15,2533	17,5363	14,9948
5	14,2637	14,8125	17,4689	14,8796
6	13,521	12,9252	17,4947	15,1472
media	14,24243333	13,97808333	17,45425	14,87273333

kernel bandwidth

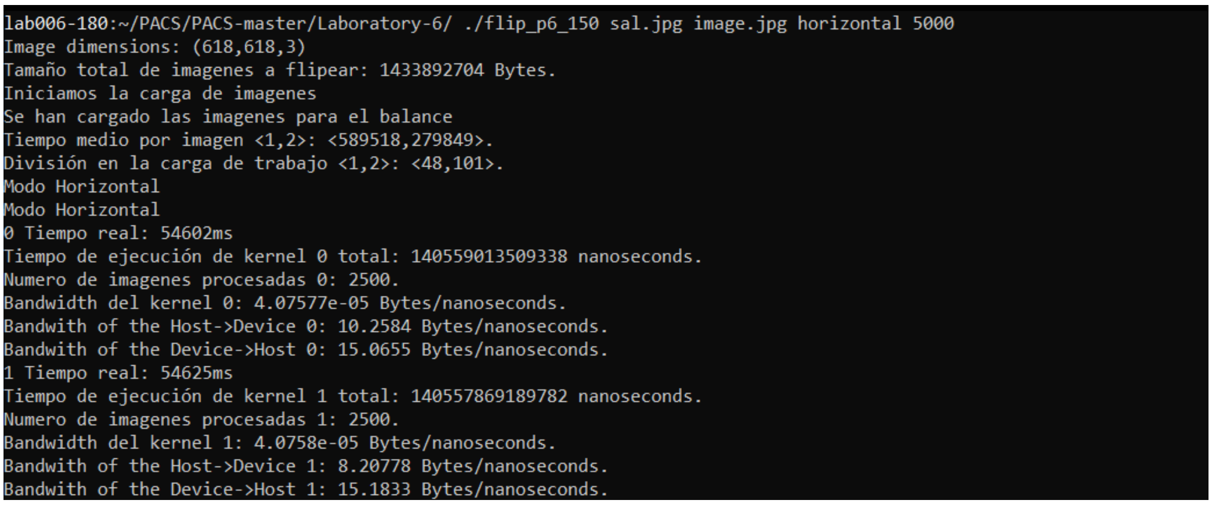
kernel (Bytes/ns)	Dev-1 Kernel time (buffer size 100)	Dev-1 kernel time (buffer size 150)	Dev-2 Kernel time (buffer size 100)	Dev-2 kernel time (buffer size 150)
1	4,06578E-05	4,07577E-05	0,000041149	0,000040758
2	4,05533E-05	4,12032E-05	4,09776E-05	4,08425E-05
3	4,05761E-05	4,06085E-05	4,08697E-05	4,08368E-05
4	4,10387E-05	4,11448E-05	4,10387E-05	4,08499E-05
5	4,07962E-05	4,08674E-05	4,10913E-05	4,08674E-05
6	4,08127E-05	4,08879E-05	4,12398E-05	4,08882E-05
media	4,07391E-05	4,09116E-05	4,1061E-05	4,08405E-05

Execution with a buffer with capacity for 100 images



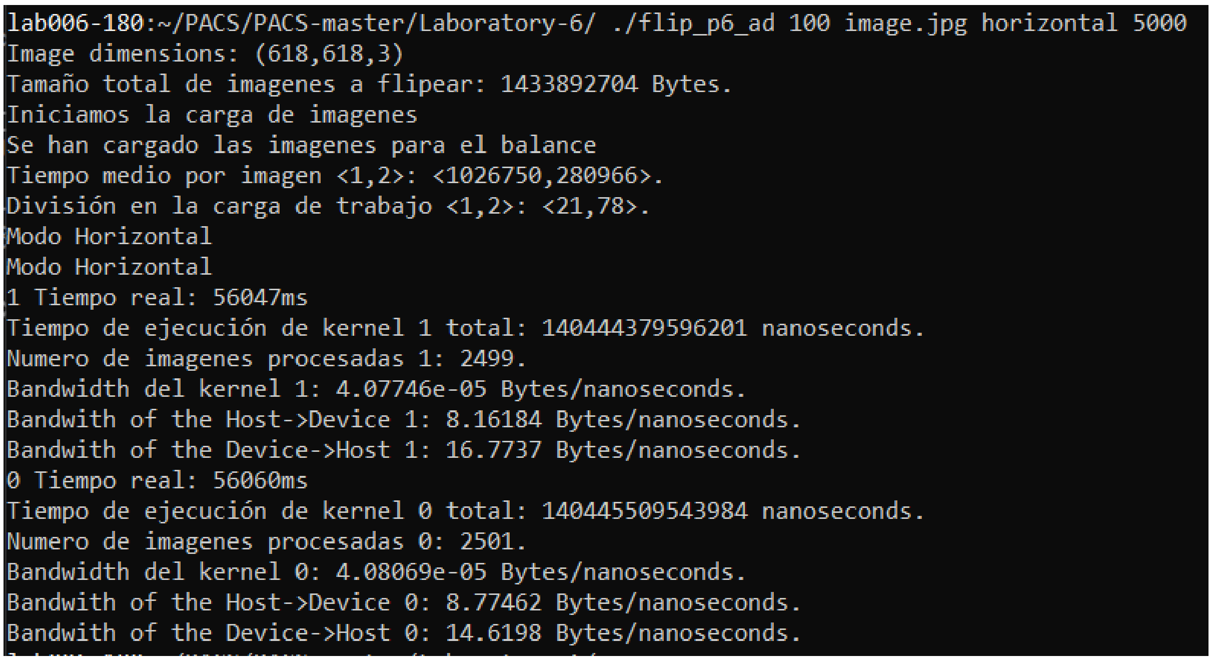
```
lab006-180:~/PACS/PACS-master/Laboratory-6/ ./flip_p6_100 sal.jpg image.jpg horizontal 5000
Image dimensions: (618,618,3)
Tamaño total de imagenes a flipear: 1433892704 Bytes.
Iniciamos la carga de imagenes
Se han cargado las imagenes para el balance
Tiempo medio por imagen <1,2>: <600874,278999>.
División en la carga de trabajo <1,2>: <31,68>.
Modo Horizontal
Modo Horizontal
0 Tiempo real: 54846ms
Tiempo de ejecución de kernel 0 total: 140058893609052 nanoseconds.
Numero de imagenes procesadas 0: 2485.
Bandwidth del kernel 0: 4.06578e-05 Bytes/nanoseconds.
Bandwith of the Host->Device 0: 9.48381 Bytes/nanoseconds.
Bandwith of the Device->Host 0: 14.9316 Bytes/nanoseconds.
1 Tiempo real: 54869ms
Tiempo de ejecución de kernel 1 total: 140057701633305 nanoseconds.
Numero de imagenes procesadas 1: 2515.
Bandwidth del kernel 1: 4.1149e-05 Bytes/nanoseconds.
Bandwith of the Host->Device 1: 8.47055 Bytes/nanoseconds.
Bandwith of the Device->Host 1: 17.7398 Bytes/nanoseconds.
```

Execution with a buffer with capacity for 150 images



```
lab006-180:~/PACS/PACS-master/Laboratory-6/ ./flip_p6_150 sal.jpg image.jpg horizontal 5000
Image dimensions: (618,618,3)
Tamaño total de imagenes a flipear: 1433892704 Bytes.
Iniciamos la carga de imagenes
Se han cargado las imagenes para el balance
Tiempo medio por imagen <1,2>: <589518,279849>.
División en la carga de trabajo <1,2>: <48,101>.
Modo Horizontal
Modo Horizontal
0 Tiempo real: 54602ms
Tiempo de ejecución de kernel 0 total: 140559013509338 nanoseconds.
Numero de imagenes procesadas 0: 2500.
Bandwidth del kernel 0: 4.07577e-05 Bytes/nanoseconds.
Bandwith of the Host->Device 0: 10.2584 Bytes/nanoseconds.
Bandwith of the Device->Host 0: 15.0655 Bytes/nanoseconds.
1 Tiempo real: 54625ms
Tiempo de ejecución de kernel 1 total: 140557869189782 nanoseconds.
Numero de imagenes procesadas 1: 2500.
Bandwidth del kernel 1: 4.0758e-05 Bytes/nanoseconds.
Bandwith of the Host->Device 1: 8.20778 Bytes/nanoseconds.
Bandwith of the Device->Host 1: 15.1833 Bytes/nanoseconds.
```

Example of execution with a buffer with the last version (configurable buffer size)



```
lab006-180:~/PACS/PACS-master/Laboratory-6/ ./flip_p6_ad 100 image.jpg horizontal 5000
Image dimensions: (618,618,3)
Tamaño total de imagenes a flipear: 1433892704 Bytes.
Iniciamos la carga de imagenes
Se han cargado las imagenes para el balance
Tiempo medio por imagen <1,2>: <1026750,280966>.
División en la carga de trabajo <1,2>: <21,78>.
Modo Horizontal
Modo Horizontal
1 Tiempo real: 56047ms
Tiempo de ejecución de kernel 1 total: 140444379596201 nanoseconds.
Numero de imagenes procesadas 1: 2499.
Bandwidth del kernel 1: 4.07746e-05 Bytes/nanoseconds.
Bandwith of the Host->Device 1: 8.16184 Bytes/nanoseconds.
Bandwith of the Device->Host 1: 16.7737 Bytes/nanoseconds.
0 Tiempo real: 56060ms
Tiempo de ejecución de kernel 0 total: 140445509543984 nanoseconds.
Numero de imagenes procesadas 0: 2501.
Bandwidth del kernel 0: 4.08069e-05 Bytes/nanoseconds.
Bandwith of the Host->Device 0: 8.77462 Bytes/nanoseconds.
Bandwith of the Device->Host 0: 14.6198 Bytes/nanoseconds.
```