MEMORIA DEL TRABAJO EN GRUPO

PL5_C

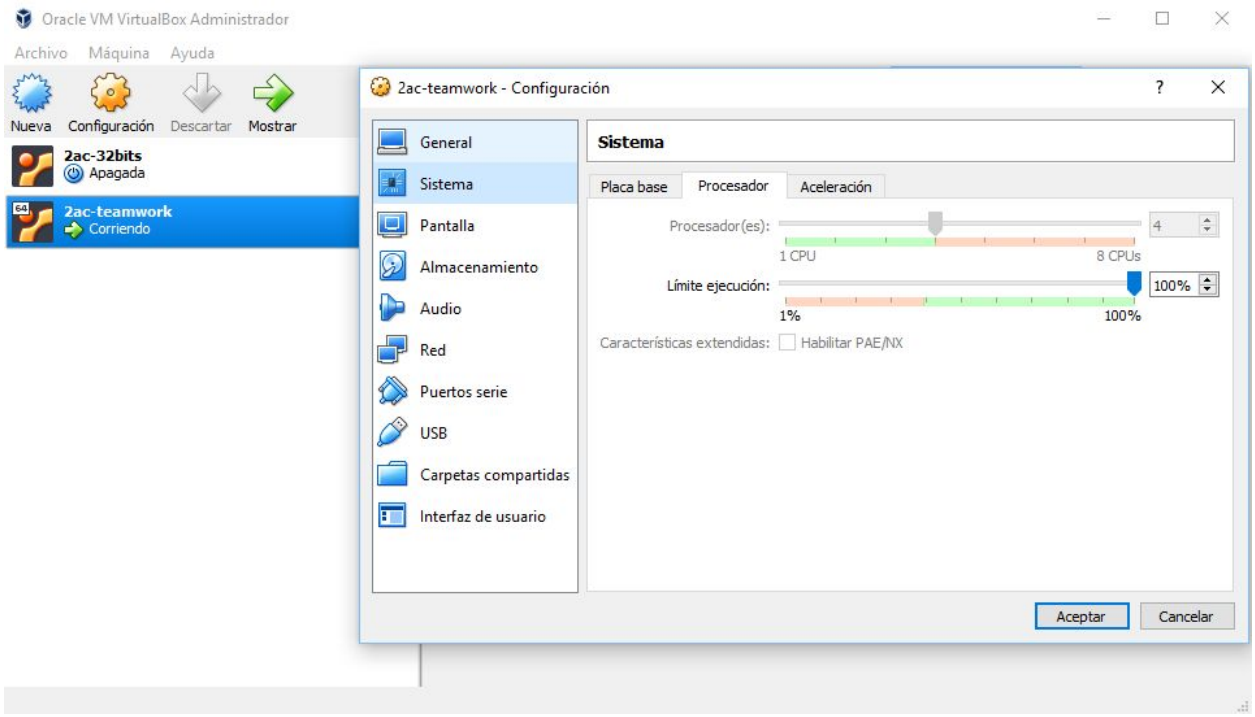
INTEGRANTES DEL GRUPO:

- López López, Ángela
- García Lavandera, Sonia.
- Cuartas Puente, Fabio.
- de Leiva Martínez, Guillermo.

REPARTO DE TRABAJO:

Para que no hubiese distinciones en el reparto de asignaciones, además de para asegurar el correcto entendimiento de los objetivos a cumplir, hemos decidido realizar el proyecto presencialmente entre los 4 miembros, de forma que todos participamos equitativamente en su realización.

PLATAFORMA DE DESARROLLO:



```
student@2ac-teamwork:~$ cat cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3199.994
cache size      : 6144 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl x
topology nonstop_tsc cpuid pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe
popcnt aes xsave avx rdrand hypervisor lahf_lm abm invpcid_single pti fsgsbase avx2
invpcid
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6399.98
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

```
processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3199.994
cache size      : 6144 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 4
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl x
topology nonstop_tsc cpuid pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe
popcnt aes xsave avx rdrand hypervisor lahf_lm abm invpcid_single pti fsgsbase avx2
invpcid
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6399.98
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

```
processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3199.994
cache size      : 6144 KB
physical id     : 0
siblings        : 4
core id         : 2
cpu cores       : 4
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl x
topology nonstop_tsc cpuid pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe
popcnt aes xsave avx rdrand hypervisor lahf_lm abm invpcid_single pti fsgsbase avx2
invpcid
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6399.98
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 60
model name      : Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
stepping        : 3
cpu MHz         : 3199.994
cache size      : 6144 KB
physical id     : 0
siblings        : 4
core id         : 3
cpu cores       : 4
apicid          : 3
initial apicid  : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl x
topology nonstop_tsc cpuid pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe
popcnt aes xsave avx rdrand hypervisor lahf_lm abm invpcid_single pti fsgsbase avx2
invpcid
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 6399.98
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

student@2ac-teamwork:~$
```

EXPLICACIÓN DE LA REALIZACIÓN DEL TRABAJO:

FASE 1: SINGLETHREAD

La primera parte consiste en la realización de un programa monohilo que ejecute un algoritmo en dos pasos, Contraste + Sepia siguiendo las indicaciones mostradas:

1) Contraste

Aumenta o disminuye el contraste de una imagen. Algoritmo:

T = porcentaje de variación del contraste [-100, 100] El valor 0 no hace nada.

$C = ((100 + T)/100)^2$

R' = R * C

G' = G * C

B' = B * C

2) Convertir a sepia el resultado

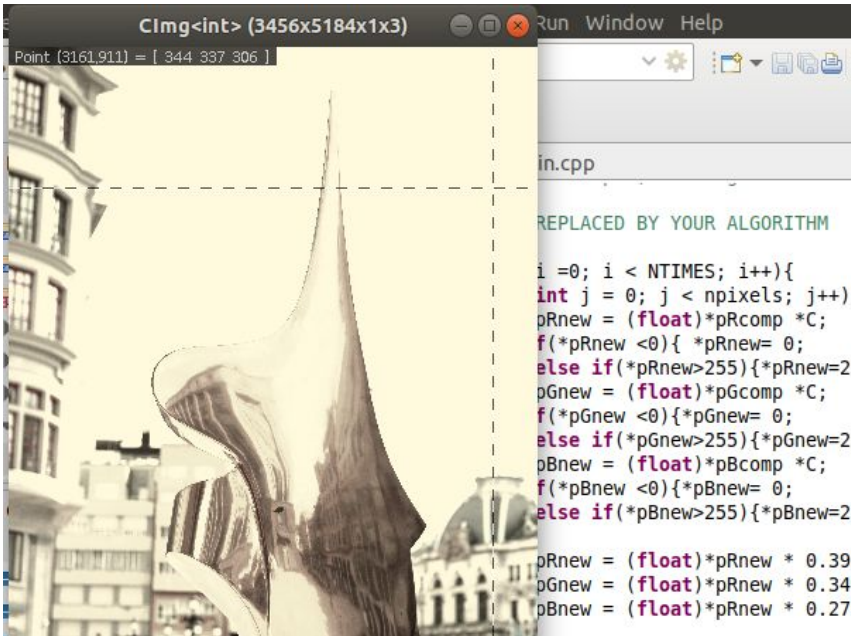
Fórmula:

R' = 0,393 * R + 0,769 * G + 0,189 * B

G' = 0,349 * R + 0,686 * G + 0,168 * B

B' = 0,272 * R + 0,534 * G + 0,131 * B

Es necesario controlar la saturación para que el resultado no se salga fuera de los límites permitidos:



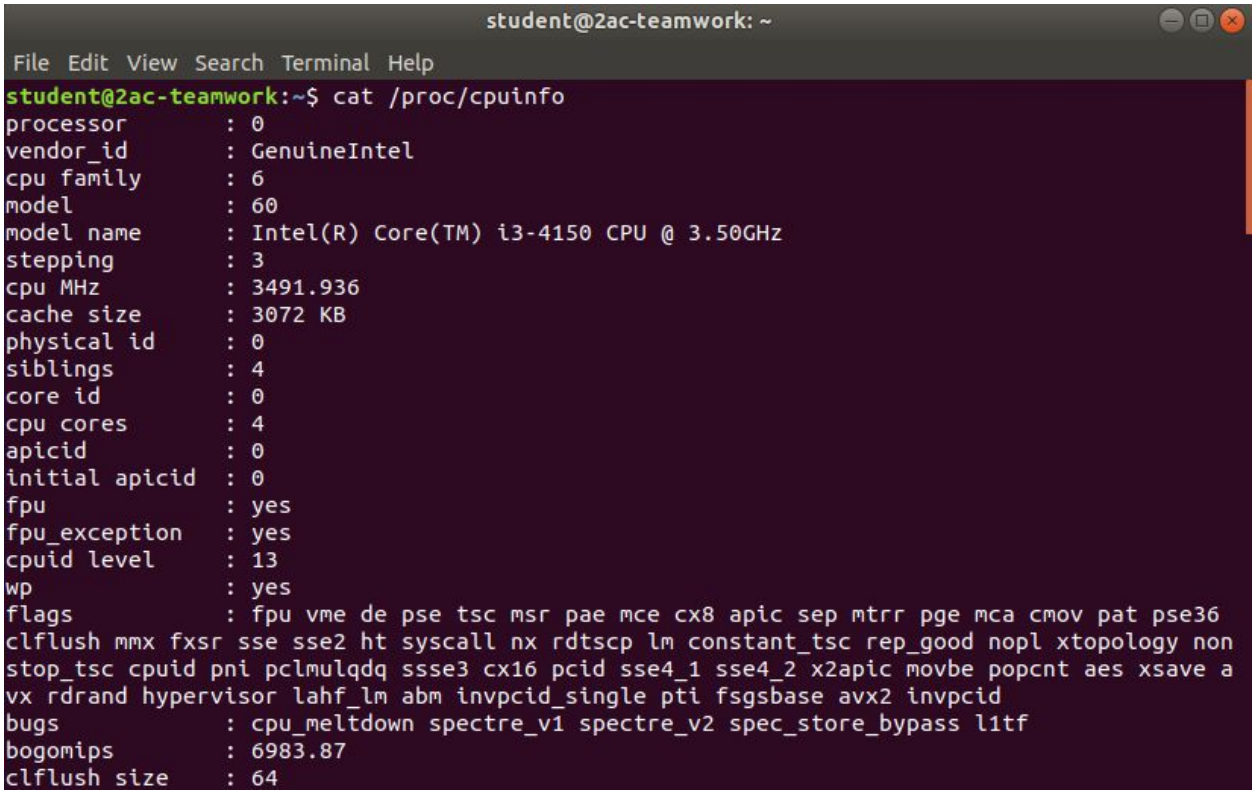
Para ello es necesario comprobarlo tras aplicar el contraste pues es una multiplicación por un número entre 100 y 100 y tras sepia, pues es una multiplicación de los componentes anteriores por componentes cuya suma es superior a 1.

Este programa debe mostrar por consola el tiempo total transcurrido entre el inicio del algoritmo hasta el final del mismo, midiendo únicamente la ejecución del procesamiento vectorial y obviando los tiempos de carga e inicialización de las imágenes, así como el tiempo necesario para guardarlas en disco.
Como la medida obtenida era un resultado demasiado pequeño, implementamos un bucle durante NTIMES veces para obtener un resultado final comprendido entre 5 y 10 segundos.

Los datos referidos a este algoritmo se pueden encontrar en la tabla de documentación bajo el nombre SINGLETHREAD.

FASE 2: SINGLETHREAD-SIMD

La segunda parte consiste en utilizar extensiones SIMD para mejorar el rendimiento. Primero se procede a ver qué extensiones soporta, ya que estas dependen del procesador:



Como nuestro grupo trabaja con instrucciones instrínsecas de 128, empleamos SSE,SSE2 y SSE3, ya que son las más modernas a las que tenemos acceso que nos proporcionan lo que necesitamos.

Esto se debe a que trabajamos con 32 bits y las más modernas traen funciones para 64 bits.

Los datos referidos a este algoritmo se pueden encontrar en la tabla de documentación bajo el nombre SINGLETHREAD-SIMD.

La aceleración de este respecto a singlethread se justifica con el uso de las instrucciones intrínsecas, ya que estas se usan en el algoritmo (ya que se necesita aplicarlo sobre varios datos a la vez):
Para ello dividimos los datos en paquetes para los componentes R,G,B de origen y de destino. Como trabajamos con 128 /32 = 4 pixels/paquete, se leen 16 componentes de memoria por iteración con *_mm_lddqu_si128.*
Dado que trabajar con int trae consigo problemas a la hora de operar, tenemos que pasar los paquetes a float para poder trabajar con ellos *_mm_cvtepi32_ps.*
Realizamos el algoritmo en dos pasos visto en singlethread, usando *_mm_mul_ps* para multiplicar, *_mm_set1_ps* como puntero a un dato, *_mm_min_ps* y *_mm_max_ps* para realizar la saturación y *_mm_add_ps* para realizar una suma entre dos valores.
Una vez realizado el algoritmo, es necesario volver a convertir los paquetes float a int para entregar el resultado pedido. Para ello usamos *_mm_cvtps_epi32.*

Como se puede observar, con el uso de estas funciones se reducen las iteraciones necesarias y por lo tanto, mejora el rendimiento.

Para poder comparar el resultado obtenido con el obtenido en singlethread, mantuvimos el mismo número de NTIMES en el bucle.

Después por debug comprobamos cuales son las funciones que se llaman en lenguaje ensamblador, las cuales son las siguientes:

vlddqu
vaddps
vminps
vbroadcastss
vmaxps
vcvtdp2ps
vmulps
vmovaps
vmovqu

Y aquí las capturas demostrando como la consola nos las muestra:

la función vaddps:

```
 186                 return (__m128) ((__v4sf)__A + (__v4sf)__B);
0000555555557c94:    vmovaps -0xe0(%rbp),%xmm0
0000555555557c9c:    vaddps  -0xd0(%rbp),%xmm0,%xmm0
0000555555557ca4:    vmovss  0x125134(%rip),%xmm1        # 0x55555567cde0
0000555555557cac:    vmovss  %xmm1,-0x4cc(%rbp)
```

la función vmulps:

```
 198                 return (__m128) ((__v4sf)__A * (__v4sf)__B);
0000555555557c74:    vmovaps -0xc0(%rbp),%xmm1
0000555555557c7c:    vmulps  -0xb0(%rbp),%xmm1,%xmm1
0000555555557c84:    vmovaps %xmm1,-0xe0(%rbp)
0000555555557c8c:    vmovaps %xmm0,-0xd0(%rbp)
```

la función vcvtdq2ps :

```
105                           bsPacket = _mm_cvtepi32_ps(tmp128iPacket3);
0000555555557930:    vmovdqa -0x40(%rbp),%xmm0
0000555555557935:    vmovaps %xmm0,-0x3a0(%rbp)
000055555555793d:    vmovdqa -0x3a0(%rbp),%xmm0
0000555555557945:    vcvtdq2ps       %xmm0,%xmm0
0000555555557949:    vmovaps %xmm0,-0x400(%rbp)
0000555555557951:    vmovss  -0x51c(%rbp),%xmm0
0000555555557959:    vmovss  %xmm0,-0x4fc(%rbp)
```

la función vmaxps:

```
 121                          rdPacket = _mm_max_ps(rdPacket, _mm_set1_ps(0.0));
0000555555557ac6:    vmovaps %xmm0,-0x3f0(%rbp)
0000555555557ace:    vmovss  0x1252fe(%rip),%xmm0        # 0x55555567cdd4
0000555555557ad6:    vmovss  %xmm0,-0x4f4(%rbp)
 894                 return __extension__ (__m128)(__v4sf){ __F, __F, __F, __F };
0000555555557ade:    vbroadcastss    -0x4f4(%rbp),%xmm0
0000555555557ae7:    vmovaps %xmm0,%xmm1
0000555555557aeb:    vmovaps -0x3e0(%rbp),%xmm0
0000555555557af3:    vmovaps %xmm0,-0x2f0(%rbp)
0000555555557afb:    vmovaps %xmm1,-0x2e0(%rbp)
 228                 return (__m128) __builtin_ia32_minps ((__v4sf)__A, (__v4sf)__B);
0000555555557b03:    vmovaps -0x2f0(%rbp),%xmm0
0000555555557b0b:    vminps  -0x2e0(%rbp),%xmm0,%xmm0
```

la función vminps:

```
120                     rdPacket = _mm_min_ps(rdPacket, _mm_set1_ps(255.0));
 0000555555557a7d:   vmovaps %xmm0,-0x3f0(%rbp)
 0000555555557a85:   vxorps  %xmm0,%xmm0,%xmm0
 0000555555557a89:   vmovss  %xmm0,-0x4f8(%rbp)
  894                return __extension__ (__m128)(__v4sf){ __F, __F, __F, __F };
 0000555555557a91:   vbroadcastss    -0x4f8(%rbp),%xmm0
 0000555555557a9a:   vmovaps %xmm0,%xmm1
 0000555555557a9e:   vmovaps -0x3f0(%rbp),%xmm0
 0000555555557aa6:   vmovaps %xmm0,-0x310(%rbp)
 0000555555557aae:   vmovaps %xmm1,-0x300(%rbp)
  234                return (__m128) __builtin_ia32_maxps ((__v4sf)__A, (__v4sf)__B);
 0000555555557ab6:   vmovaps -0x310(%rbp),%xmm0
 0000555555557abe:   vmaxps  -0x300(%rbp),%xmm0,%xmm0
```

y por último la función load que usa vmovaps y vmovqu:

```
 97                      tmp128iPacket1 = _mm_loadu_si128((__m128i*) pRcomp);
 00005555555578a1:   vmovaps %xmm0,-0x440(%rbp)
 00005555555578a9:   mov     -0x4b8(%rbp),%rax
 00005555555578b0:   mov     %rax,-0x460(%rbp)
 00005555555578b7:   mov     -0x460(%rbp),%rax
 00005555555578be:   vmovdqu (%rax),%xmm0
```

FASE 3: MULTITHREAD

La tercera parte consiste en desarrollar una versión multihilo del programa singlethread.
En nuestro caso *Hyperthreading* **no esta activado** ya que solo ejecuta 1 hilo por procesador, lo que a la hora de obtener los hilos necesarios aplicaremos la fórmula 4 procesadores x 4 nucleos = 16 hilos

```
CPU(s):                  4
On-line CPU(s) list:     0-3
Thread(s) per core:      1
Core(s) per socket:      4
Socket(s):               1
```

Advanced Technologies

Intel® Turbo Boost Technology ‡	2.0
Intel® vPro™ Platform Eligibility ‡	Yes
Intel® Hyper-Threading Technology ‡	No
Intel® Virtualization Technology (VT-x) ‡	Yes
Intel® Virtualization Technology for Directed I/O (VT-d) ‡	Yes
Intel® VT-x with Extended Page Tables (EPT) ‡	Yes
Intel® TSX-NI	No
Intel® 64 ‡	Yes
Instruction Set	64-bit
Instruction Set Extensions	Intel® SSE4.1, Intel® SSE4.2, Intel® AVX2
Intel® My WiFi Technology	Yes
Idle States	Yes
Enhanced Intel SpeedStep® Technology	Yes
Thermal Monitoring Technologies	Yes

El uso de hilos permite descomponer el algoritmo y mejorar el rendimiento, ya que cada hilo puede ejecutarse de forma concurrente con otros hilos. De esta forma se puede ejecutar distintas partes del algoritmo en paralelo.

Para poder comparar el resultado obtenido con el obtenido en singlethread, mantuvimos el mismo número de NTIMES en el bucle.

Los datos referidos a este algoritmo se pueden encontrar en la tabla de documentación bajo el nombre MULTITHREAD.