ASIGNATURA Computación de altas prestaciones

Práctica 1 Parte0-S1: Instalar cluster Rocks utilizando Maquina Virtuales

EJERCICIOS de la práctica1-parte0-sección1 (15%)

Ejercicio 1: Pare y reinicie el cluster de manera ordenada. Enumere los pasos que ha realizado para conseguirlo.

Paso 1 - Entrar en modo SuperUsuario. Comando: su

Paso 2 - Reiniciar todos los nodos del clúster. Comando: rocks run host reboot

Paso 3 - Reiniciar el frontend. Comando: reboot

Ejercicio 2: Suponga que uno de los nodos ha fallado. Elimine el nodo de las bases de datos de rocks y reinstale uno nuevo en su lugar manteniendo el nombre del antiguo, aunque no se mantenga necesariamente su IP. Enumere los pasos que ha realizado para conseguirlo.

Suponemos fallo en nodo 1:

Paso 1 - Eliminar el nodo del clúster y mostrar el estado final de este.
Comando: insert-ethers --remove compute-0-1; rocks list host; rocks sync config

```
[root@clusterlab7 Desktop]# insert-ethers --remove compute-0-1; rocks list host; rocks sync config
Removed node compute-0-1
HOST         MEMBERSHIP CPUS RACK RANK RUNACTION INSTALLACTION
clusterlab7: Frontend   1    0    0    os        install
compute-0-0: Compute    1    0    0    os        install
```

Ilustración 1. Eliminación de un nodo en un clúster

Paso 2 - Instalar otro nodo con el mismo nombre.
Comando: insert-ether --hostname compute-0-1

Paso 3 - Comprobamos la reinserción del nodo
Comando: rocks list host

```
[root@clusterlab7 Desktop]# insert-ethers --hostname compute-0-1
[root@clusterlab7 Desktop]# insert-ethers --hostname compute-0-1
Node compute-0-1 already exists.
Select a different hostname, cabinet and/or rank value.
[root@clusterlab7 Desktop]# ls
[root@clusterlab7 Desktop]# rocks list host
HOST         MEMBERSHIP CPUS RACK RANK RUNACTION INSTALLACTION
clusterlab7: Frontend   1    0    0    os        install
compute-0-0: Compute    1    0    0    os        install
compute-0-1: Compute    1    0    1    os        install
```

Ilustración 2. Reinserción del nodo al clúster

A veces, en virtualización, al hacer la reinserción de un nodo este se queda en reinicio constante. Para solucionarlo, simplemente lo apagamos y lo reinsertamos manualmente con el comando “insert-ethers” en el frontend.

Ejercicio 3: Realice un script que compruebe el estado de los nodos de computo y sea capaz de devolver el porcentaje de nodos que están activos en el cluster. Adicionalmente la salida debe indicar para cada nodo, su nombre, su dirección IP y su estado. Considere que un nodo no esta activo si la columna `states` indica `au`.

```
#!/bin/bash

numNodes=$(rocks run host "hostname" | wc -l)
noActiveNodes=$(qstat -f | grep "compute" | grep au | wc -l)
let activeNodes=$numNodes-$noActiveNodes

echo "Numero de nodos = $numNodes"
echo "Numero de nodos no activos = $noActiveNodes"
echo "Porcentaje de nodos activos = $activeNodes/$numNodes"
echo
echo "Listado de Nodos"

for node in $(rocks run host "hostname" | cut -d. -f1)
do
        ip=$(rocks run host $node "hostname -i")
        active=$(qstat -f | grep $node | awk '{if ($6=="au") {print "NO ACTIVO"}
else {print "ACTIVO"}}')
        echo -e $node'\t'$ip'\t'$active
done
```

Ilustración 3. Script para comprobación de estado de los nodos de cómputo

Ejercicio 4: Realice un script que añada usuarios al cluster rocks y devuelva como salida el nombre y el password de cada usuario. El nombre debe ser usuBDxx, donde xx se incrementa de 01 a 20 y utilice una política de passwords razonable.
¿Cómo puede verificar cuantos usuarios existen en el cluster y que nombre tienen?

```
#!/bin/bash

for i in {01..20}
do
        newUser=$(echo usuBD$i)
        userPasswd=$(echo $newUser$RANDOM)
        useradd $newUser -p userPasswd
        echo usuario $newUser:$userPasswd >> lista20usuarios.txt
done
```

Ilustración 4. Script para la creación de usuarios

La cantidad de usuarios se comprueba con: ***cat /etc/passwd | wc -l***

El nombre de los usuarios se comprueba con: ***cat /etc/passwd | cut -d: -f1***

Ejercicio 5: Realice las siguientes pruebas de conexión entre el las MV.
5.1.- Conexión desde el anfitrión al cluster (sería el equivalente a un acceso en remoto);
$ ssh bigdata@192.168.182.100

```
[eps@clusterlab7 Desktop]$ ssh eps@172.16.238.100
Warning: Permanently added '172.16.238.100' (RSA) to the list of known hosts.
Last login: Fri Sep 23 15:40:28 2022 from localhost.localdomain
Rocks 6.2 (SideWinder)
Profile built 17:02 22-Sep-2022

Kickstarted 19:08 22-Sep-2022
```

Ilustración 5. Conexión al clúster desde el anfitrión

5.2.- Desde la sesión anterior conectarse a un nodo de computo:
$ ssh compute-0-0

```
[eps@clusterlab7 ~]$ ssh compute-0-0
Warning: untrusted X11 forwarding setup failed: xauth key data not generated
Warning: No xauth data; using fake authentication data for X11 forwarding.
Rocks Compute Node
Rocks 6.2 (SideWinder)
Profile built 15:29 06-Oct-2022

Kickstarted 18:11 06-Oct-2022
```

Ilustración 6. Conexión a un nodo de cómputo desde el clúster

5.3.- ¿Qué sudera si intentamos la conexión desde el anfitrión a los nodos de computo?
$ ssh bigdata@10.11.12.252

Como se puede observar en la imagen siguiente, no ocurre ningún problema al intentar acceder a los nodos de cómputo desde el anfitrión.

```
[eps@clusterlab7 Desktop]$ ssh eps@10.1.2.254
Last login: Fri Oct  7 17:05:13 2022 from clusterlab7.local
Rocks Compute Node
Rocks 6.2 (SideWinder)
Profile built 15:29 06-Oct-2022

Kickstarted 18:11 06-Oct-2022
[eps@compute-0-0 ~]$ exit
logout
Connection to 10.1.2.254 closed.
```

Ilustración 7. Intento de conexión a nodo de cómputo desde el anfitrión

Práctica 1 - parte0-S2: Ejecución y Planificación de tareas en cluster Rocks

EJERCICIOS: Práctica 1 – Parte0 – Sección2 (15%)

Ejercicio 6: Con un editor cree un fichero con el siguiente contenido
compute-0-0
compute-0-1
compute-0-2

y denomínelo **maquinasMPI.txt**

Pruebe el siguiente comando:

$ /opt/openmpi/bin/mpirun -np 10 –machinefile maquinasMPI.txt hostname

y explique como varía el resultado respecto al comando:

/opt/openmpi/bin/mpirun -np 10 hostname

Responda a las siguientes preguntas:

- ¿Se ha ejecutado algún proceso en el frontend?
  No

```
[eps@clusterlab7 Desktop]$ gedit maquinasMPI.txt
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 10 -machinefile maquinasMPI.txt hostname
compute-0-1.local
compute-0-0.local
compute-0-1.local
compute-0-0.local
compute-0-1.local
compute-0-1.local
compute-0-0.local
compute-0-0.local
compute-0-1.local
compute-0-0.local
```

Ilustración 8. Ejecución de mpirun con clúster

- ¿Cómo se puede ejecutar parte de los procesos en el frontend?
  Añadiendo el localhost al archivo "maquinasMPI.txt"

```
[eps@clusterlab7 Desktop]$ gedit maquinasMPI.txt
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 10 -machinefile maquinasMPI.txt hostname
clusterlab7.ii.uam.es
clusterlab7.ii.uam.es
compute-0-1.local
compute-0-1.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-1.local
clusterlab7.ii.uam.es
compute-0-0.local
```

Ilustración 9. Ejecución de mpirun con clúster, frontend incluido

- Varíe el número de procesos e intente deducir el reparto de tareas que se utiliza. **Intenta repartir una tarea a cada máquina usando Round Robin, de manera que queda equilibrado.**

```
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 100 -machinefile maquinasMPI.txt hostname | sort | uniq -c
     33 clusterlab7.ii.uam.es
     34 compute-0-0.local
     33 compute-0-1.local
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 112 -machinefile maquinasMPI.txt hostname | sort | uniq -c
     37 clusterlab7.ii.uam.es
     38 compute-0-0.local
     37 compute-0-1.local
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 102 -machinefile maquinasMPI.txt hostname | sort | uniq -c
     34 clusterlab7.ii.uam.es
     34 compute-0-0.local
     34 compute-0-1.local
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 99 -machinefile maquinasMPI.txt hostname | sort | uniq -c
     33 clusterlab7.ii.uam.es
     33 compute-0-0.local
     33 compute-0-1.local
```

Ilustración 10. Comprobación de reparto de tareas en el clúster

Ejercicio 7: Compile los ejemplos de mpi disponibles en `/opt/mpi-tests/`e indique como funcionan.

Reparte las 6 ejecuciones entre las máquinas disponibles, de modo que cada máquina adopta varias ejecuciones, enviando y recibiendo mensajes identificándose como distintos procesos.

```
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 6 -machinefile maquinasMPI.txt /opt/mpi-tests/bin/mpi-ring
Process 2 on clusterlab7.ii.uam.es
Process 3 on compute-0-0.local
Process 4 on compute-0-1.local
Process 5 on clusterlab7.ii.uam.es
Process 1 on compute-0-1.local
Process 0 on compute-0-0.local
Process 3 on compute-0-0.local:successfully sent (1048576) bytes to id (4)
Process 1 on compute-0-1.local:successfully sent (1048576) bytes to id (2)
Process 4 on compute-0-1.local:successfully sent (1048576) bytes to id (5)
Process 4 on compute-0-1.local:successfully received (1048576) bytes from id (3)
Process 2 on clusterlab7.ii.uam.es:successfully sent (1048576) bytes to id (3)
Process 0 on compute-0-0.local:successfully sent (1048576) bytes to id (1)
Process 2 on clusterlab7.ii.uam.es:successfully received (1048576) bytes from id (1)
Process 1 on compute-0-1.local:successfully received (1048576) bytes from id (0)
Process 5 on clusterlab7.ii.uam.es:successfully sent (1048576) bytes to id (0)
Process 5 on clusterlab7.ii.uam.es:successfully received (1048576) bytes from id (4)
Process 3 on compute-0-0.local:successfully received (1048576) bytes from id (2)
Process 0 on compute-0-0.local:successfully received (1048576) bytes from id (5)
[eps@clusterlab7 Desktop]$ /opt/openmpi/bin/mpirun -np 6 -machinefile maquinasMPI.txt /opt/mpi-tests/bin/mpi-verify
Process 5 on clusterlab7.ii.uam.es
Process 4 on compute-0-1.local
Process 2 on clusterlab7.ii.uam.es
Process 3 on compute-0-0.local
Process 1 on compute-0-1.local
Process 0 on compute-0-0.local
```

Ilustración 11. Ejecución de varios procesos dentro del clúster

Ejercicio 8: Crear una cola denominada colapares.q basándose en la cola all.q que tenga solo los nodos con nombres compute-0-x, siendo x par. Compruebe su funcionamiento.

Indique los comandos utilizados para su creación y el proceso para comprobar su funcionamiento.

Generamos un archivo txt con la configuración del grupo *allhosts*.
Comando: qconf -shgrp @allhosts > computePares.txt

Accedemos a la configuración nueva para cambiar el nombre del grupo y los nodos que pertenecerán.
Comando: vi computePares.txt

Creamos un nuevo grupo con la nueva configuración.
Comando: qconf -Ahgrp computePares.txt

Comprobamos que el grupo se creó correctamente.
Comando: qconf -shgrpl

Generamos un archivo txt con la configuración de la cola *all.q*.
Comando: qconf -sq all.q > colasPares.txt

Accedemos a la configuración nueva para cambiar el nombre de la cola, el nombre del grupo al que pertenecerá y los nodos que pertenecerán.
Comando: vi colasPares.txt

Creamos una nueva cola con la nueva configuración.
Comando: qconf -Aq colasPares.txt

Comprobamos que la cola se creó correctamente.
Comando: qconf -sql

Comprobamos que podemos acceder a la configuración de la nueva cola desde la cola.
Comando: qconf -sq colaspares.q

```
[root@clusterlab7 Desktop]# qconf -shgrp @allhosts > computePares.txt
[root@clusterlab7 Desktop]# vi computePares.txt
[root@clusterlab7 Desktop]# qconf -Ahgrp computePares.txt
root@clusterlab7.local added "@computePares" to host group list
[root@clusterlab7 Desktop]# qconf -shgrpl
@MyHostGroup
@allhosts
@computePares
[root@clusterlab7 Desktop]# qconf -sq all.q > colasPares.txt
[root@clusterlab7 Desktop]# vi colasPares.txt
[root@clusterlab7 Desktop]# qconf -Aq colasPares.txt
root@clusterlab7.local added "colaspares.q" to cluster queue list
[root@clusterlab7 Desktop]# qconf -sql
MyCola.q
all.q
colaspares.q
[root@clusterlab7 Desktop]# qconf -sq colasPares.txt
No cluster queue or queue instance matches the phrase "colasPares.txt"
[root@clusterlab7 Desktop]# qconf -sq colaspares.q
qname                 colaspares.q
hostlist              @computePares
seq_no                0
load_thresholds       np_load_avg=1.75
suspend_thresholds    NONE
nsuspend              1
suspend_interval      00:05:00
priority              0
min_cpu_interval      00:05:00
processors            UNDEFINED
qtype                 BATCH INTERACTIVE
ckpt_list             NONE
pe_list               make mpich mpi orte
rerun                 FALSE
slots                 1,[compute-0-0.local=1]
tmpdir                /tmp
```

Ilustración 12. Creación de la cola "colapares.q"

Ejercicio 9: Cree nuevas colas de acuerdo con los criterios que le parezcan significativos, por ejemplo, cola1core para máquinas con un solo procesador y cola2core para máquinas con dos procesadores. Para ello reinstale el nodo 1 con 2 cores y cree una nuevo nodo compute-0-3 con 2 cores. Realice pruebas para comprobar el funcionamiento de las colas creadas.

Indique los comandos utilizados para su creación y el proceso para comprobar su funcionamiento.

Los comandos son los mismos que en el ejercicio anterior, solo cambia el contenido de los archivos de configuración: nombres de grupos y colas y los nodos que pertenecen a cada una.

```
[root@clusterlab7 Desktop]# qconf -shgrp @allhosts > core1.txt
[root@clusterlab7 Desktop]# vi core1.txt
[root@clusterlab7 Desktop]# qconf -Ahgrp core1.txt
root@clusterlab7.local added "@core1" to host group list
[root@clusterlab7 Desktop]# qconf -sq all.q > cola1core.txt
[root@clusterlab7 Desktop]# vi cola1core.txt
[root@clusterlab7 Desktop]# qconf -Ah cola1core.txt
error: invalid option argument "-Ah"
Usage: qconf -help
[root@clusterlab7 Desktop]# qconf -Aq cola1core.txt
root@clusterlab7.local added "cola1core.q" to cluster queue list
[root@clusterlab7 Desktop]# qconf -shgrp @core1
group_name @core1
hostlist compute-0-0.local compute-0-2.local
[root@clusterlab7 Desktop]# qconf -sq cola1core.q
qname                 cola1core.q
hostlist              @core1
seq_no                0
load_thresholds       np_load_avg=1.75
suspend_thresholds    NONE
nsuspend              1
suspend_interval      00:05:00
priority              0
min_cpu_interval      00:05:00
processors            UNDEFINED
qtype                 BATCH INTERACTIVE
ckpt_list             NONE
pe_list               make mpich mpi orte
rerun                 FALSE
slots                 1,[compute-0-0.local=1],[compute-0-2.local=1]
tmpdir                /tmp
```

```
[root@clusterlab7 Desktop]# qconf -shgrp @allhosts > core2.txt
[root@clusterlab7 Desktop]# vi core2.txt
[root@clusterlab7 Desktop]# qconf -Ahgrp core2.txt
root@clusterlab7.local added "@core2" to host group list
[root@clusterlab7 Desktop]# qconf -sq all.q > cola2core.txt
[root@clusterlab7 Desktop]# vi cola2core.txt
[root@clusterlab7 Desktop]# qconf -Aq cola2core.txt
root@clusterlab7.local added "cola2core.q" to cluster queue list
[root@clusterlab7 Desktop]# qconf -shgrp @core2
group_name @core2
hostlist compute-0-1.local compute-0-3.local
[root@clusterlab7 Desktop]# qconf -sq cola2core.q
qname                 cola2core.q
hostlist              @core2
seq_no                0
load_thresholds       np_load_avg=1.75
suspend_thresholds    NONE
nsuspend              1
suspend_interval      00:05:00
priority              0
min_cpu_interval      00:05:00
processors            UNDEFINED
qtype                 BATCH INTERACTIVE
ckpt_list             NONE
pe_list               make mpich mpi orte
rerun                 FALSE
slots                 1,[compute-0-1.local=1],[compute-0-3.local=1]
tmpdir                /tmp
```

Ilustración 13. Creación de colas "cola1core.q" y "cola2core.q"

Práctica1 Parte 1 (35%)
Vector processing and SIMD

You must write a report answering the questions proposed in each exercise, plus the requested files. Submit a zip file through Moodle. Check submission date in Moodle (deadline is until 11:59 pm of that date).

- Exercise 1:
  - Identify your CPU model and list the supported SIMD instructions.

```
e420552@6A-7-6-7:~$ cat /proc/cpuinfo | grep "model name" | uniq
model name      : Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz
```

Ilustración 14. Modelo de CPU del Laboratorio

```
e420552@6A-7-6-7:~$ gcc -c -Q -march=native --help=target | grep "enabled" | awk '{print $1}' | tr '\n' ' '; echo
-m128bit-long-double -m64 -m80387 -mabm -madx -maes -malign-stringops -mavx -mavx2 -mbmi -mbmi2 -mcrc32 -mcx16 -mf16c
 -mfancy-math-387 -mfma -mfp-ret-in-387 -mfsgsbase -mfxsr -mglibc -mhard-float -mhle -mieee-fp -mlong-double-80 -mlzc
nt -mmmx -mmovbe -mmwait -mpclmul -mpopcnt -mprfchw -mpush-args -mrdrnd -mrdseed -mred-zone -mrtm -msahf -msse -msse2
 -msse3 -msse4 -msse4.1 -msse4.2 -mssse3 -mstv -mtls-direct-seg-refs -mvzeroupper -mxsave -mxsaveopt
```

Ilustración 15. Instrucciones SIMD soportadas por la CPU del Laboratorio

  - Explain the main differences between both assembly codes (vectorized and non-vectorized) focused on the SIMD instructions generated by the compiler.

La principal diferencia se encuentra en los bucles en los que la versión nativa ejecuta literalmente lo que se encuentra en el *.c*, es decir, únicamente hace las dos asignaciones en cada iteración, como se puede ver evidenciado en las 2048 iteraciones que hace el bucle en el código ensamblador. Mientras que la versión vectorizada emplea vectores de 32 bytes para mejorar la eficiencia del programa, ejecutando una misma instrucción sobre el conjunto de datos con los vectores de 32 bytes.

- Exercise 2:
  - Provide the source code of *simple2_intrinsics.c* after the vectorization of the loops. Explain how you have carried out the vectorization of the code.

Hemos seguido en gran medida un esquema similar a la vectorización empleada en los bucles de computación empleando vectores de 256 bits, es por ello que el bucle en vez de realizar k iteraciones realiza k/4 pues sabiendo que cada double ocupa 8 bytes que son 64 bits 256/64=4 es decir en estos vectores nos caben 4 dobles lo que nos permite reducir las iteraciones del bucle, dentro del bucle, cada iteración inicializa los vectores __m256d vb = {i, i+1, i+2, i+3}; para después guardarlos en array de dobles mediante las instrucciones _mm256_store_pd(&a[i], va); _mm256_store_pd(&b[i], vb); (también se ha aplicado un sistema para que sea funcional con números no múltiplos de 4 simplemente añade ceros hasta llegar al múltiplo)

- Compare the execution time for different values of NUMBER_OF_TRIALS: from 100.000 to 1.000.000 in steps of 100.000. Plot the results in a graph. Discuss the results.

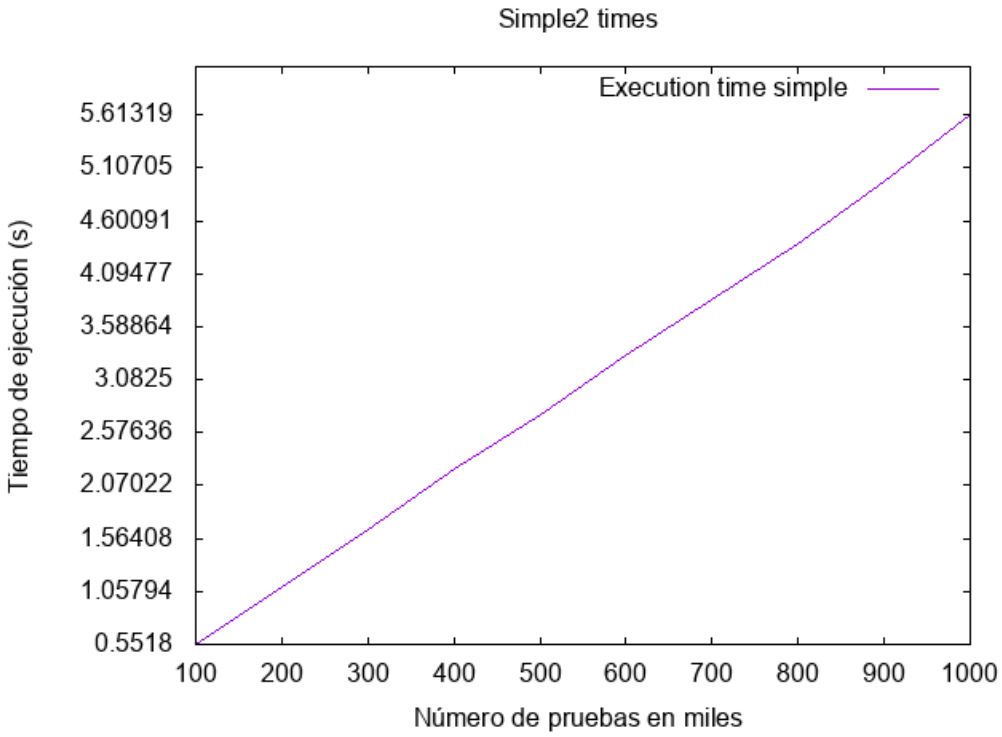

Ilustración 16. Tiempos de ejecución de simple2.c

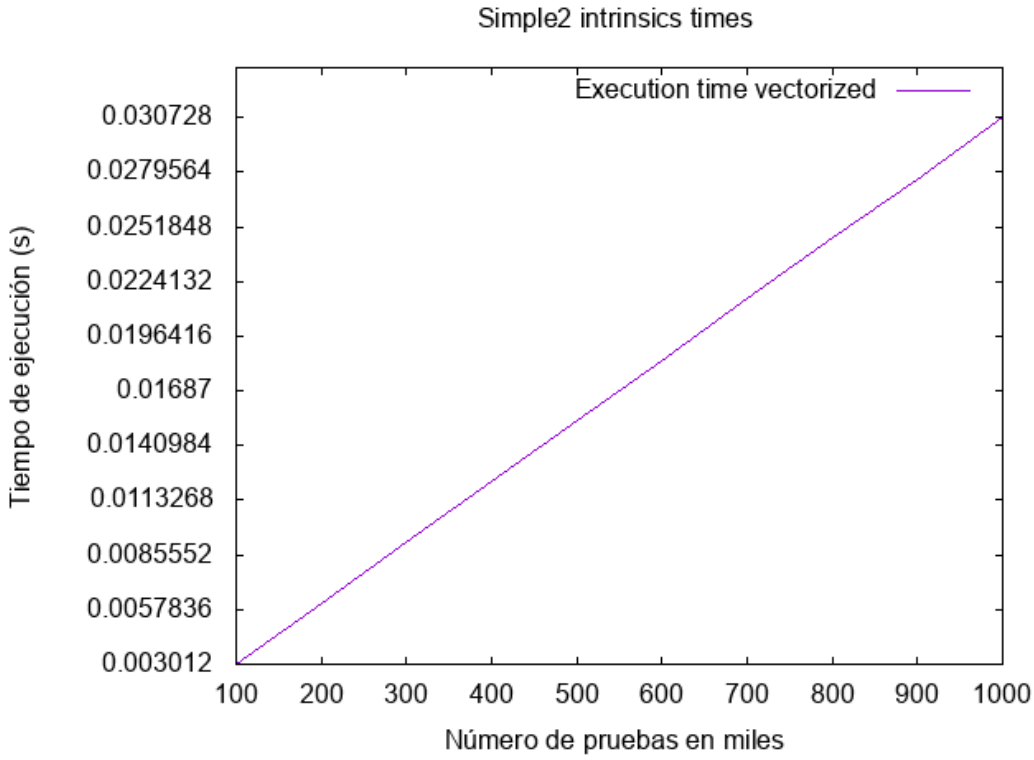

Ilustración 17. Tiempos de ejecución de simple2_intrinsics.c

Como se puede apreciar la ejecución de ambos sigue una evolución en línea prácticamente recta con pendiente constante sin embargo la versión vectorizada logra unos resultados de computación bastante superiores aproximadamente de 200 veces más rápido (unas 182 veces más rápido) que la versión simple.

A continuación dejamos una gráfica comparativa en la que el eje y de la izquierda refleja los tiempos de la ejecución del programa simple mientras que el eje de la derecha refleja los tiempo de la versión vectorizada, como se puede ver y se intuía de las anteriores gráficas, la forma es muy similar así como su crecimiento sin embargo una comienza en aproximadamente 0.6 segundos, mientras que la vectorizada comienza la gráfica en 0.003 segundos lo que indica que la versión vectorizada es 200 veces más rápido.

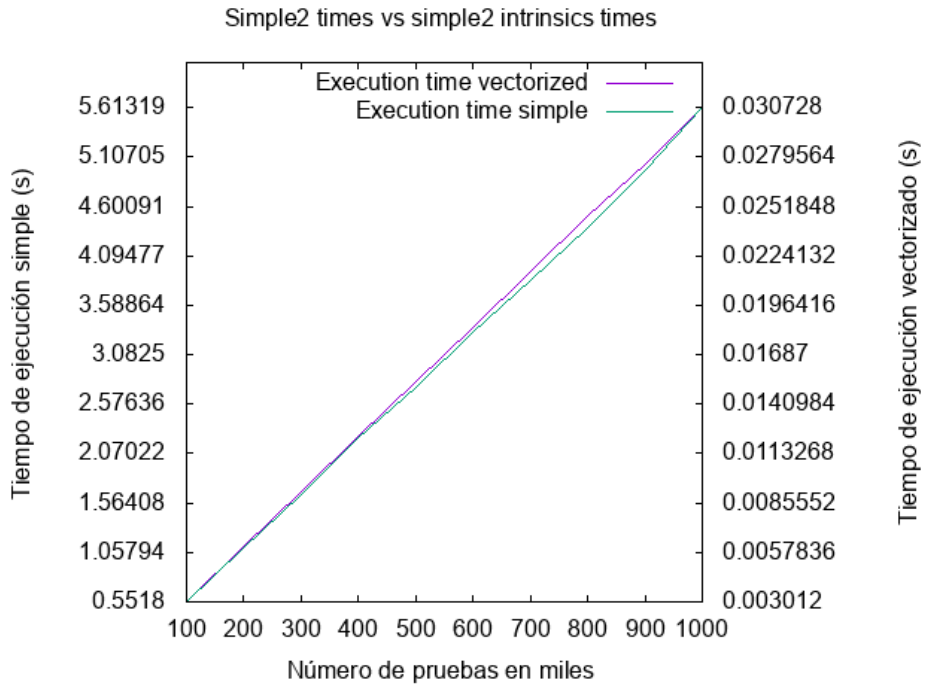

Ilustración 18. Comparación de tiempos de ejecución entre simple2.c y simple2_intrinsics.c

- Exercise 3:
    - The program includes two loops. The first loop (indicated as Loop 0) iterates over the arguments applying the algorithm to each of them. The second loop (indicated as Loop 1) computes the grey scale algorithm. Is this loop optimal to be vectorized? Why?

El programa básicamente modifica la tonalidad de rojo, verde y azul de cada pixel de la/s imagen/es recibida/s por argumento. Para ello, en un bucle reserva memoria en un array unidimensional de tamaño *altura * anchura* de la imagen, recorre la imagen original pixel a pixel y modifica las tonalidades de rojo, verde y azul, multiplicando el valor original de cada uno por un cierto coeficiente. Finalmente, guarda el resultado en modo jpg.

- Provide the source code of the auto-vectorized version of the code. Explain the changes in the code to help the compiler to vectorize the loop.

Este bucle interno no es óptimo para ser vectorizado, por el hecho de que el acceso a datos se hace muy ineficiente. Ineficiente porque accede multiplicando el valor de la columna, por la anchura de la imagen y le suma la fila en la que se encuentra.

El array que contiene la imagen modificada es unidimensional, esto hace que usar el cálculo descrito antes conlleve ir dando saltos por el array; al igual que en el acceso a la imagen original, utiliza el offset para moverse por la memoria que contiene la imagen, pero lo hace a través de saltos en la memoria y no de corrido.

- Provide the source code after manually vectorizing the code. Explain your solution.

La solución más viable es hacer que el acceso a las posiciones de memoria sea continuo, sin saltos, y esto se consigue accediendo a través de la multiplicación de la fila por la altura y sumar la columna. De esta forma, se recorre la memoria de seguido, como si se moviera a la casilla del lado cada vez.

Estos cambios han de hacerse tanto en el índice del array de la imagen modificada como en el cálculo del offset de la función auxiliar *getRGB()*.

- Fill in a table with time and speedup results compared to the original version and auto-vectorized version for images of different resolutions (SD, HD, FHD, UHD-4k, UHD-8k). You must include a column with the fps at which the program would process. Discuss the results.

Calidad de Imagen	Versión Original		Versión Auto-vectorizada		SpeedUp
	Tiempo (seg)	FPS	Tiempo (seg)	FPS	
SD	0.0038	259.8	0.0031	318.2	1.2258
HD	0.0159	62.6	0.0118	84.7	1.3475
FHD	0.0379	26.4	0.0265	37.7	1.4302
4k	0.1700	5.9	0.1013	9.9	1.6782
8k	0.8426	1.2	0.4007	2.5	2.1028

Tabla 1. Tiempos y aceleraciones de greyScale

Como podemos observar, la versión actualizada genera tiempos menores como era de esperar. También se puede apreciar que afecta de una forma cada vez más notoria según aumenta la calidad de la imagen. Esto último es lógico pues tiene que recorrer cada vez más pixeles, su trabajo es mayor y los tiempos crecen a la par. Así que, si un código es más eficiente que otro, su curva de tiempos será menos pronunciada.

Esto también se puede comprobar a través de la aceleración. Puesto que el trabajo que aumenta es en la parte paralelizable, las aceleraciones se acusan cada vez más.

```
__m256 values = _mm256_setr_ps(0.2989f, 0.5870f, 0.1140f, 0.0, 0.2989f, 0.5870f, 0.1140f, 0.0);
__m256 res;
__m128i grey;
__m256i offset = _mm256_setr_epi32(0, 4, 1, 5, 0, 0, 0, 0);

for (int j = 0; j < height; j += 4)
{
    for (int i = 0; i < width; i += 4)
    {
        __m128i datal = _mm_loadl_epi64((__m128i *)(rgb_image + (i + width * j) * 4));
        __m128i datah = _mm_loadl_epi64((__m128i *)(rgb_image + (i + width * j) * 4 + 8));
        __m256i data32l = _mm256_cvtepu8_epi32(datal);
        __m256i data32h = _mm256_cvtepu8_epi32(datah);
        __m256 datafl = _mm256_cvtepi32_ps(data32l);
        __m256 datafh = _mm256_cvtepi32_ps(data32h);
        __m256 finall = _mm256_mul_ps(datafl, values);
        __m256 finalh = _mm256_mul_ps(datafh, values);

        res = _mm256_hadd_ps(finall, finalh);
        res = _mm256_floor_ps(_mm256_hadd_ps(res, res));
        res = _mm256_permutevar8x32_ps(res, offset);
        grey = _mm_cvtps_epi32(_mm256_extractf128_ps(res, 0));

        __m128i bperm = _mm_setr_epi8(0, 1*4, 2*4, 3*4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
        grey = _mm_shuffle_epi8(grey, bperm);
        uint32_t *tt = (uint32_t *)&grey;
        *((uint32_t *)(&grey_image[j * width + i])) = *tt;
    }
}
```

Ilustración 19. Código modificado para la vectorización manual del bucle en greyScale.c

Intrinsics usados:

- **_mm256_setr_ps:** crea un vector de 32 bits a través de 8 floats en orden inverso.
- **_mm256_setr_epi32:** crea un vector de 32 bits a través de 8 integers en orden inverso.
- **_mm_loadl_epi64**: carga un vector 64 bits de tipo integer desde memoria.
- **_mm256_cvtepu8_epi32**: extiende un vector con unsigned integers de 8 bits a uno con integers de 32 bits.
- **_mm256_cvtepi32_ps**: convierte un vector con integers de 32 bits a floats de 32 bits.
- **_mm256_mul_ps**: multiplica dos vectores de floats de 32 bits entrada a entrada.
- **_mm256_hadd_ps:** suma horizontalmente por pares las posiciones de los vectores.
- **_mm256_floor_ps:** redondea los valores del vector de su valor float a integer.
- **_mm256_permutevar8x32_ps:** mezcla los valores de los vectores.
- **_mm256_extractf128_ps:** extrae 128 bits (4 paquetes de floats de 32 bits) dependiendo del segundo argumento (0: parte baja, 1: parte alta).
- **_mm_cvtps_epi32:** convierte floats de 32 bits a enteros de 32 bits.
- **_mm_setr_epi8:** crea un vector de 128 bits con integers de 8 bits en orden inverso.
- **_mm_shuffle_epi8:** mezcla los integers de 8 bits según el vector de 128 bits.

Idea de la vectorización manual:

Como no podemos usar AVX-512 y tenemos que usar AVX2 (256 bits), tenemos que transformar la idea de 512 al “doble” de 256.

La idea general es usar los vectores para coger pixeles de cuatro en cuatro, pero como usamos AVX2, hemos de usars dos vectores con dos pixeles cada uno por iteración. Estos vectores los extendemos a enteros de 32 bits y luego los pasamos a floats para poder multiplicarlos por el vector común que modifica los valores RGB a gris.

Después hacemos el proceso contrario para conseguir vectores que nos interesen: usamos dos veces el intrinsic que suma horizontalmente y redondeamos, porque de esta manera se consiguen sumar los 4 bytes de cada pixel, aunque el resultado queda desordenado. Es por esto por lo que usamos otro intrinsic para reordenarlos. Una vez reordenados, recogemos la parte del vector que nos interesa y lo preparamos para guardar en la posición de la imagen modificada que le toca a cada pixel.