

ASIGNATURA Computación de altas prestaciones

Práctica 1 Parte0-S1 : Instalar cluster Rocks utilizando Máquinas 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.

To reboot the cluster the correct way we need to follow the next steps:

First, we execute the following command to power off the nodes of the cluster. This should be done by root.

```
$ rocks run host "poweroff"
```

The next step will be to shutdown the frontend of the cluster. To achieve this, we run the following command:

```
$ shutdown -H now
```

Now the cluster is completely powered off. To turn it on again, we must follow the next steps:

Open the frontend (in the Virtual Machine)

Open the cluster nodes (in the Virtual Machine)

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.

Computación de altas prestaciones HPC

A node has failed and we must remove it and install a new one with the same name. To achieve this goal we must follow the next steps:

First, we must power off the Virtual Machine with the computer. This will allow us to manipulate the nodes while they are not running.

Secondly, we remove the corrupt node. We can achieve this by using the following commands:

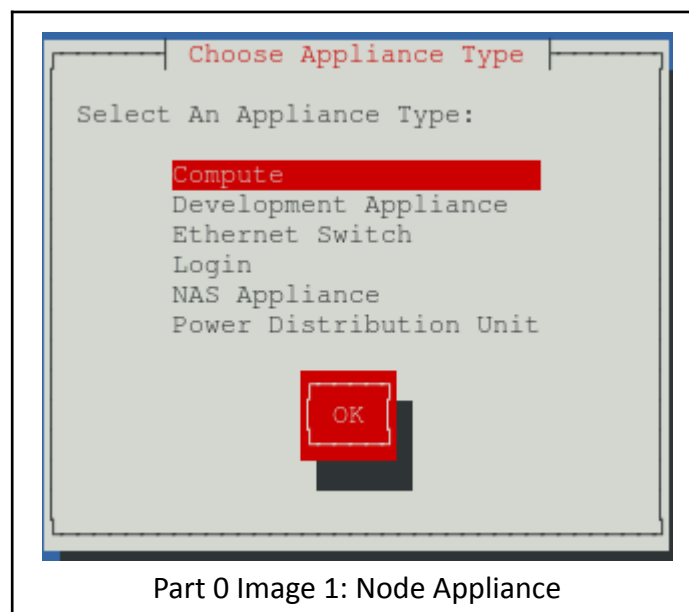
```
insert-ethers --remove compute-0-10;  
rocks list host;  
rocks sync config
```

Where compute-0-10 is the name of the node we want to remove. This will remove the node and synchronize the cluster to work without it.

No the node has been removed, the next step will be to create a new node with the same name as the previously deleted. We can do this by using the following command:

```
insert-ethers --hostname compute-0-10;
```

A menu will show up with different options for the Appliance Type of the new node. We need a Compute Node, so we select the compute option and press OK.



Finally we power on the Virtual Machine with the deleted node, where it will be reinstalled by itself with the same name.

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 está activo si la columna `states` indica `au`.

To check the nodes we use the “rock list host” command. From here we can extract the active nodes information using the following script:

```

nodos=`rocks list host | grep compute | wc -l`
echo "Numero de nodos = $nodos"
nonodos=`qstat -f | grep NA | wc -l`
echo "Numero de nodos no activos = $nonodos"
porc=$((echo "scale=2; (1-($nonodos/$nodos))*100" | bc -l))
echo "Porcentaje de nodos activos = $porc %"
echo ""
echo "Listado de nodos"
echo "`rocks list host | grep compute | grep -v au | awk '{print \$1 \"\$2\" ACTIVO}'`"

list=`rocks list host | grep -v "host name" | awk '{print \$1}'`
# get the ip address of each node
for node in $list
do
ip=`rocks list host interface $node | grep -v "host name" | awk '{print \$2}'`
# check if the node is active
active=`rocks list host interface $node | grep -v "host name" | awk '{print \$3}'`
# output the results to a file
echo "$node,$ip,$active" >> ips.csv
done

```

In order to also add the IP to the output we arrived at the conclusion that we could use `nslookup` of the host name but unfortunately while we were doing exercise 9 we connected the cluster to one of the lab PCs and the frontend stopped working so we never got a chance to prove it.

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 cuántos usuarios existen en el cluster y que nombre tienen?

To create new users for the cluster we created the following script:

```

for i in `seq 1 20`;
do
    if [ $i -lt 10 ];
    then
        user="usuBD0$i"
    else
        user="usuBD$i"
    fi
    password=`openssl rand -base64 32`
    echo "usuario $user:$password" >> list20users.txt
    useradd -m -p $password $user
done
rocks sync users

```

As the script shows, new users and their corresponding passwords will be saved in a text file called list20users.txt. After running the script, the file looked like this:

```

usuario    usuBD01:VqInypjAYL07timWEqHm7ZIXII4FUIGmjAFA3IpeU10=
usuario    usuBD02:VASglRolRftNrEk9ePtQVTnno4/t9LxlKUTQ5wXXzlo=
usuario    usuBD03:N/DG/rdp4LdpkMrFx81o/Vh/jGKIpcy7cFX6n7hUwJE=
usuario    usuBD04:tVdZUPsNgKluFNMqEjT/PPzKhKLFAqPwigXdCyK/zDE=
usuario    usuBD05:b2nvI75OG6Z48rXmVytebpeIICzEvKWLZpgnHpa/OgQ=
usuario    usuBD06:Lvn1qe1J2nJs j6MMPDXSQxv jLz9B+UAYOKQN9aYHHxs=
usuario    usuBD07:GJtqnDSl3JMZh6+9fEAVA3X1PUgwh6YTIN8PVMWg j4=
usuario    usuBD08:hE4cgkeC0U2c6mCQtufhlXqq5aViI0EmAA6587I1b/U=
usuario    usuBD09:ndvb7x8js4RI3uIyUVxcP54f+eSfzws2/lz8pkh7RY4=
usuario    usuBD10:mh3uK4Bu+BNkNVouUZs4RkUlwzqY8XKjK4gEDYJbRRw=
usuario    usuBD11:Hdc0jXzxIvSPd2jogTuqE2/bU5wJuTF7omTTP5Qu8dw=
usuario    usuBD12:itIE4tlx0iWCAp3h9gQ3u9dbL2GOLiy jIvk6kcyENqY=
usuario    usuBD13:cVOn2ybjb1WwnXZaME3zfRghbl+bqzmUMj3Ej bEyGms=
usuario    usuBD14:FVaGGfIpJHzGuy9+ZH5bRjWWwlge6U+M53WyP1YgI+Q=
usuario    usuBD15:+0vLsf8s1Xhsx+V3PyFzJ5D4pW67R74EPLW8G4rRW6E=
usuario    usuBD16:5by21kA3hykz+/3Ep/OAeBw57fctkGJThCvOUTHwSc=
usuario    usuBD17:ESOWBah8gyKUwyJrA80JUv8oLiIs8Bt7pw6RHElnWhA=
usuario    usuBD18:AuPbMO/ksclj7Q2ibBLgLCqErUGfEY2KqAxyciIJQhk=
usuario    usuBD19:Acua3CTcwBCYYwhwO98YEpj00YYnsNts0r74YXxNrwk=
usuario    usuBD20:2T3Ty1NT+7+uxzIgfIs/dg7CfczLNYMCmzYR4oCJ6LU=

```

To check the existing users of the cluster and their corresponding names we can use the following command:

```
$ cat /etc/passwd
```

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);

We connect to the cluster from the host by using the following command:

```
$ ssh bigdata@192.168.182.100
```

After connecting we achieve the following result:

```

[root@clusterlab05 Desktop]# ssh guilledani@172.16.238.100
Warning: Permanently added '172.16.238.100' (RSA) to the list of known hosts.
Password:
Rocks 6.2 (SideWinder)
Profile built 12:15 06-Oct-2022

Kickstarted 14:24 06-Oct-2022

```

Now we can use the cluster from the host.

5.2.- Desde la sesión anterior conectarse a un nodo de computo:

```
$ ssh compute-0-0
```

From the new cluster session we can create a new connection to one of the cluster nodes, achieving the following result:

```
[guilledani@clusterlab05 ~]$ ssh compute-0-6
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 12:52 06-Oct-2022

Kickstarted 14:58 06-Oct-2022
```

5.3.- ¿Qué sucederá si intentamos la conexión desde el anfitrión a los nodos de computo?

\$ ssh bigdata@10.11.12.252

We finally try to access one of the nodes directly from the host, achieving the following result:

```
[root@clusterlab05 Desktop]# ssh guilledani@10.11.12.248
Warning: Permanently added '10.11.12.248' (RSA) to the list of known hosts.
guilledani@10.11.12.248's password:
Last login: Thu Oct 6 16:56:44 2022 from clusterlab05.local
Rocks Compute Node
Rocks 6.2 (SideWinder)
Profile built 12:52 06-Oct-2022

Kickstarted 14:58 06-Oct-2022
```

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 cómo varía el resultado respecto al comando:

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

```
[root@compute-0-0 ~]# /opt/openmpi/bin/mpirun -np 10 hostname
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
compute-0-0.local
[root@compute-0-0 ~]# /opt/openmpi/bin/mpirun -np 10 -machinefile maquinasMPI.txt hostname
compute-0-0.local
compute-0-0.local
compute-0-2.local
compute-0-2.local
compute-0-1.local
compute-0-1.local
compute-0-2.local
compute-0-1.local
compute-0-0.local
compute-0-0.local
[root@compute-0-0 ~]#
```

Responda a las siguientes preguntas:

- **¿Se ha ejecutado algún proceso en el frontend?**

No process has been executed in the frontend, because as we can see all the hostname commands show that processes have been only executed on compute nodes.

- **¿Cómo se puede ejecutar parte de los procesos en el frontend?**

```
[root@compute-0-0 ~]# rocks list host
HOST                MEMBERSHIP CPUS RACK RANK RUNACTION INSTALLACTION
clusterlab05: Frontend 1 0 0 os install
compute-0-2: Compute 1 0 2 os install
compute-0-0: Compute 1 0 3 os install
compute-0-1: Compute 1 0 4 os install
[root@compute-0-0 ~]# nano maquinasMPI.txt
[root@compute-0-0 ~]# /opt/openmpi/bin/mpirun -np 10 -machinefile maquinasMPI.t:
t hostname
compute-0-0.local
compute-0-0.local
compute-0-2.local
compute-0-2.local
compute-0-1.local
compute-0-0.local
compute-0-1.local
compute-0-1.local
clusterlab05.ii.uam.es
clusterlab05.ii.uam.es
[root@compute-0-0 ~]#
```

In order to execute some process in the frontend, we checked the frontend host name which in our case was clusterlab05 and we introduced the hostname in `maquinasMPI.txt` so that it does not only execute in the nodes but also in the frontend.

- Varíe el número de procesos e intente deducir el reparto de tareas que se utiliza.

Looking at the variation in how processes are distributed between the nodes, we could assume there is not a clear distribution such as Round-Robin. On the other hand, we can confirm there is a balance in the assignment of the tasks.

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

```
[root@compute-0-0 ~]# /opt/openmpi/bin/mpirun -np 6 -machinefile maquinasMPI.txt /opt/mpi-tests/bin/mpi-ring
Process 4 on compute-0-0.local
Process 5 on compute-0-1.local
Process 3 on clusterlab05.ii.uam.es
Process 2 on compute-0-2.local
Process 0 on compute-0-0.local
Process 1 on compute-0-1.local
Process 2 on compute-0-2.local:successfully sent (1048576) bytes to id (3)
Process 2 on compute-0-2.local:successfully received (1048576) bytes from id (1)
Process 1 on compute-0-1.local:successfully sent (1048576) bytes to id (2)
Process 3 on clusterlab05.ii.uam.es:successfully sent (1048576) bytes to id (4)
Process 3 on clusterlab05.ii.uam.es:successfully received (1048576) bytes from id (2)
Process 4 on compute-0-0.local:successfully sent (1048576) bytes to id (5)
Process 1 on compute-0-1.local:successfully received (1048576) bytes from id (0)
Process 4 on compute-0-0.local:successfully received (1048576) bytes from id (3)
Process 0 on compute-0-0.local:successfully sent (1048576) bytes to id (1)
Process 5 on compute-0-1.local:successfully sent (1048576) bytes to id (0)
Process 5 on compute-0-1.local:successfully received (1048576) bytes from id (4)
Process 0 on compute-0-0.local:successfully received (1048576) bytes from id (5)
```

As we can observe, the mpi test-ring sends and receives data between the different nodes used. This test shows communication between nodes.

```
[root@compute-0-0 ~]# /opt/openmpi/bin/mpirun -np 6 -machinefile maquinasMPI.txt /opt/mpi-tests/bin/mpi-verify
Process 4 on compute-0-0.local
Process 2 on compute-0-2.local
Process 5 on compute-0-1.local
Process 3 on clusterlab05.ii.uam.es
Process 0 on compute-0-0.local
Process 1 on compute-0-1.local
```

On the other hand, the mpi test-verify verifies all the nodes are working correctly by assigning a process to each one of them.

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.

To list the existing queues, we use the command:

```
qconf -sql
```

This shows only one queue is available at the moment (all.q). To create a new queue called colapares, we use the commands:

First, to create all the configuration for the new queue we copy the configuration from the previously existing one:

```
qconf -sq all.q > ./colapares.txt
```

Then, we dump on a text file all the possible hosts for the queue:

```
qconf -shgrp @allhosts > ./myHosts.txt
```

We modify the myHosts.txt file so the hostlist only has the even hosts (compute-0-0 & compute-0-2). We also modify the group name from @allhosts to @myhosts.

The following commands to add the new host list:

```
su
qconf -Ahgrp ./myHosts.txt
```

Next, we check that the new hostlist named @myhosts is available with the following command:

```
qconf -shgrpl
```

On the colapares.txt we change the qname by our name, the hostlist by our hostlist and we remove the corresponding slot.

Finally, we add the queue to the queues managed by SGE via the following command:

```
qconf -Aq ./colapares.txt
```

To check the new task was added to the correct queue, we use the qsub command with the -q colapares.q . We can also use qstat -f and see that our new queue is correctly configured. (in the last two rows of the displayed table)

```
[root@clusterlab05 ~]# qstat -f
```

queue name	qtype	resv/used/tot.	load_avg	arch	state

all.q@compute-0-0.local	BIP	0/0/1	0.00	linux-x64	

all.q@compute-0-1.local	BIP	0/0/1	0.00	linux-x64	

all.q@compute-0-2.local	BIP	0/0/1	0.00	linux-x64	

colapares.q@compute-0-0.local	BIP	0/0/1	0.00	linux-x64	

colapares.q@compute-0-2.local	BIP	0/0/1	0.00	linux-x64	

Ejercicio 9: Cree nuevas colas de acuerdo a 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.

In order to create the two queues, we follow the same procedure as in exercise 8:

First, we copy the all.q queue:

```
qconf -sq all.q > ./cola1core.txt  
qconf -sq all.q > ./cola2core.txt
```

Then we copy the hosts:

```
qconf -shgrp @allhosts > ./1cores.txt  
qconf -shgrp @allhosts > ./2cores.txt
```

We modify the 1cores.txt to have only compute-0-0 and compute-0-2 (The single core nodes) We also modify the 2cores.txt to have only compute-0-1 and compute-0-3 (The dual core nodes).

```
su  
qconf -Ahgrp ./cola1core.txt  
su  
qconf -Ahgrp ./cola2core.txt
```

Change on cola1core.txt the name of the queue to cola1core.q, the host to @1cores and the slots to the corresponding nodes.

Change on cola2core.txt the name of the queue to cola2core.q, the host to @2cores and the slots to the corresponding nodes.

```
qconf -Aq ./cola1core.txt  
qconf -Aq ./cola2core.txt
```

use the qsub -f command to see that the queue is ready.

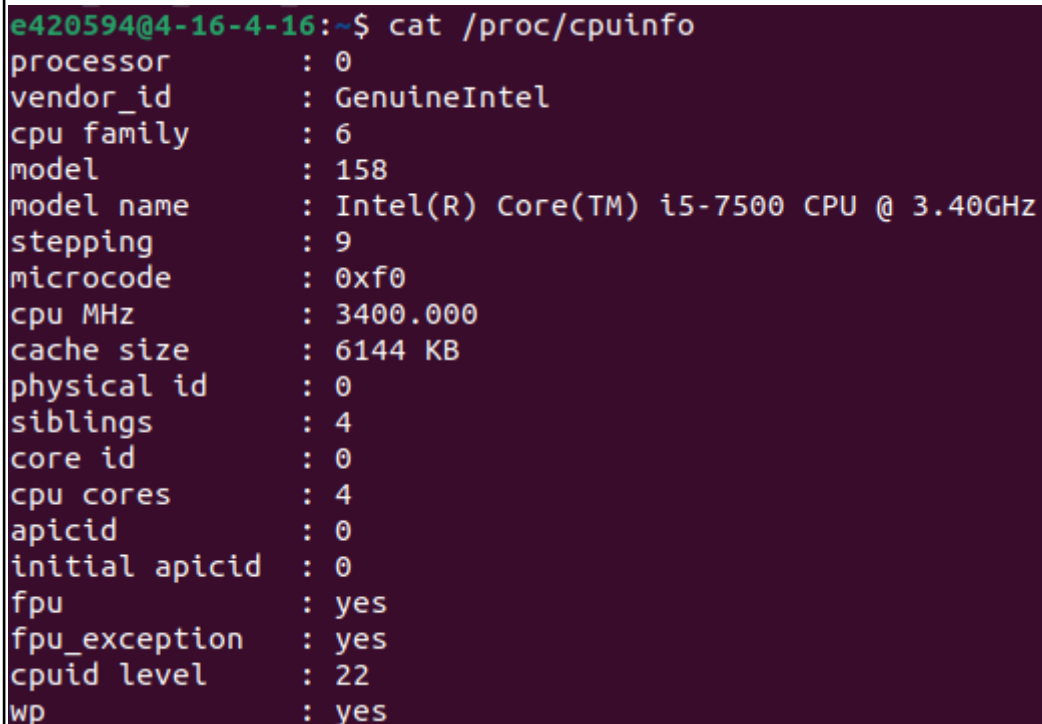
Práctica1 Parte 1 (35%) Vector processing and SIMD

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.

To identify the CPU model we can introduce the command `cat/proc/cpuinfo` and we obtain the following:



```
e420594@4-16-4-16:~$ cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 158
model name     : Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz
stepping       : 9
microcode      : 0xf0
cpu MHz        : 3400.000
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    : 22
wp             : yes
```

Part 1 Image 1: CPU info

Where in the model name we can find the model of the CPU which is "Intel® Core(TM) i5-7500 CPU @ 3.40GHz".

As to the SIMD instructions, with the previous command we can also obtain them in the flags part:

```
lags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
at pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdt
cp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_ts
cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdb
fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer a
s xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_singl
pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase
tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt inte
_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_a
t_window hwp_epp md_clear flush_l1d arch_capabilities
```

Part 1 Image 2: Flags & compatible registers

Where we need to focus only on the ones starting with “mmx”, “sse” and “avx” because they are the SIMD supported instructions.

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

When we observe both assembly codes we can spot a few differences:

- The non vectorized code is shorter than the vectorized one. This is because the vectorized code does not use loops hence it becomes larger.
- As we can observe, the code with no vectorization has simpler instructions with less vector registers (such as xmm0, xmm1, ymm, etc...)
- The code with vectorization has more complex instructions such as cvtdq2pd

● **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.

We approached the vectorization of the loops using intrinsics by trying to vectorize the sums done in the different loops to approach a faster result. To achieve this we used the following intrinsics:

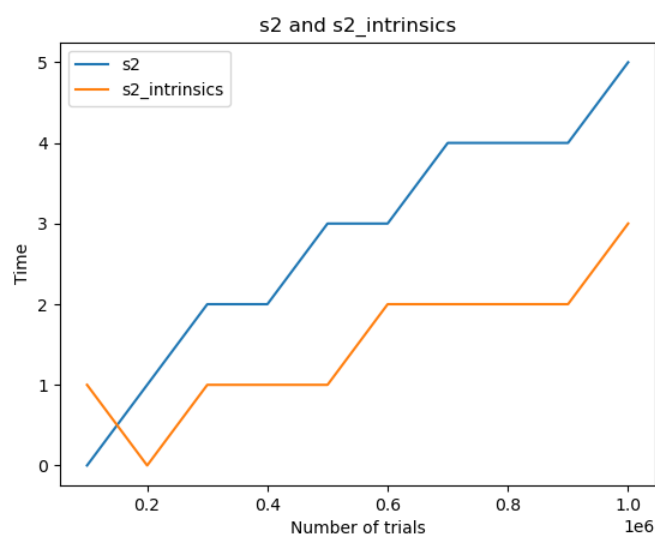
```
/* Populate A and B arrays using intrinsics */  
for (i = 0; i < ARRAY_SIZE; i += 4)  
{  
    // Initialize b vector as __m256d b = {0, 1, 2, 3}.  
    __m256d bi = _mm256_set_pd(0, 1, 2, 3);  
    // Similar idea for a.  
    __m256d ai = _mm256_set_pd(1, 2, 3, 4);  
    // Add the i constant to each element of a and b.  
    ai = _mm256_add_pd(ai, _mm256_set_pd(i,i,i,i));  
    bi = _mm256_add_pd(bi, _mm256_set_pd(i,i,i,i));  
    // Store the result in the array.  
    _mm256_store_pd(&a[i], ai);  
    _mm256_store_pd(&b[i], bi);  
}
```

Part 1 Image 3: Vectorization intrinsics

The rest of the source code is included in a separate file.

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.

A graph was plotted using a different number of trials to check the difference between the normal and the intrinsics versions. The results were the following:



Part 1 Image 4: s2 comparison plot

This shows how intrinsics times are less than no vectorizing times. This is due to the optimizations done by the intrinsics, where computations are vectorized and done faster.

- **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?

This loop is not optimal to be vectorized because it makes unnecessary jumps to obtain the data from the cache. For it to be optimal to be vectorized the information that we fetch from the cache should be consecutive so we can make more than one operation simultaneously.

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.

The source code of the auto-vectorized version would be the same as the provided code. The changes we would have to make to the code to help the compiler vectorize the loop would be switching the order of the loops of the computations. This would make is to that the data that we try to access is inline in the cache hence allowing us to vectorize (because we are able to access groups of data instead of one by one).

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

To manually vectorize the code we decided to vectorize the second loop of the computation. To achieve this, we needed to work with vectors and use registers compatible with the computer we were using. We achieved this by using the AVX2 registers.

To load each of the vectors representing the colors we needed to separate them into two 8 bytes vectors, low and high. Then we used intrinsics as expected to compute the results. Finally we reorganized the vectors to achieve the final result to save the image.

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.

We ran three different programs:

Intrinsics: without -O3 flag and with our manual vectorization.

Auto-Vectorized: with -O3 flag and automatic vectorization.

No Vectorized: without -O3 flag and without vectorization.

Image	Intrinsics Time	Auto-Vectorized Time	No Vectorization Time	Speedup Intrinsics/No Vec	Speedup Intrinsics/Auto Vec	FPS (with intrinsics)
8k.jpg	1.343888	0.283226	2.042638	0.6579178494	4.744931609	0.7441096282
4k.jpg	0.338852	0.079124	0.410573	0.8253148648	4.282543855	2.951140911
FHD.jpg	0.090366	0.02037	0.102461	0.8819550853	4.43622975	11.06610893
HD.jpg	0.041671	0.010599	0.050594	0.8236352137	3.931597321	23.99750426
SD.jpg	0.013144	0.004412	0.014187	0.9264819906	2.979147779	76.08034084

The results obtained, as expected, show faster times using intrinsics than the ones obtained without optimizing the code. This is due to the intrinsics used. The bigger the image gets the more noticeable the difference is, suggesting an exponential growth. On the other hand, auto vectorization times are faster than the manual ones. This happens due to the compiler optimizing the code in better ways and with more specific optimizations (with the -O3 tag).

We can observe the results graphically with the following plot:

Intrinsics Time, Auto-Vectorized Time and No Vectorization Time

