**ASIGNATURA Computación de altas prestaciones**

**Práctica 1 Parte0-S1 : Instalar cluster Rocks utilizando Máquina Virtuales**

**DESTACAR:** Aunque se muestre parte de código, dejamos entregados junto a esta memoria los archivos implementados.

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

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

Para parar cualquier nodo del cluster se puede realizar de las siguientes manera:

Ejecutando el comando “*rocks run host compute-0-X poweroff*”, para parar un nodo concreto del cluster.

Ejecutar el comando “*rocks run host poweroff*”, para parar todos los nodos del cluster.

De forma análoga realizaremos lo mismo para reiniciarlos:

Con el comando “*rocks run host compute-0-X reboot*”, reiniciamos un nodo concreto del cluster.

Con el comando “*rocks run host reboot*”, reiniciamos todos los nodos del cluster.

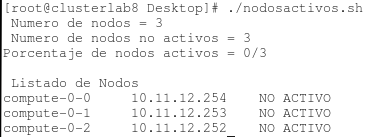
**Ejercicio 2:**  **Suponga que uno de los nodos ha fallado. Elimine el nodo de las bases de datos de rocks y reinstale uno nuevo en su lugar manteniendo el nombre del antiguo, aunque no se mantenga necesariamente su IP. Enumere los pasos que ha realizado para conseguirlo.**

En primer lugar, eliminamos el nodo del cluster, en este caso si queremos eliminar el nodo0 deberíamos ejecutar el comando “insert-ethers –remove compute-0-0; rocks list host; rocks sync config”. Una vez eliminado el nodo del cluster si queremos volver a instalar un nuevo nodo pero con el mismo nombre que el del nodo eliminado, en este caso el nodo0, deberíamos ejecutar el comando “insert-ethers –hostname compute-0-0”, este comando permitiría que el siguiente nodo que se instale tenga el nombre indicado pero su IP sería diferente.

**Ejercicio 3:** **Realice un script que compruebe el estado de los nodos de cómputo 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.**

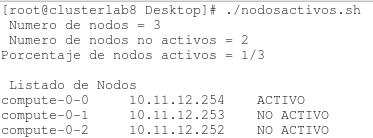
A continuación se mostrará la salida de ejecutar el script. En concreto se mostrarán 2 imágenes, una en la que ninguno de los nodos se encuentran activos y otra en la que el nodo0 si se encuentra activo.

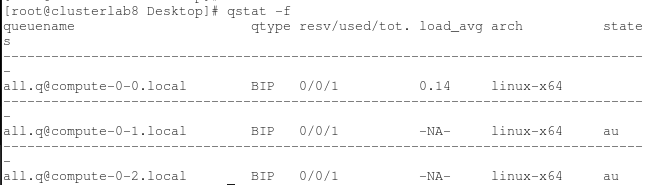
* Todos los nodos inactivos junto con evidencia de ello:





* El nodo0 activo:

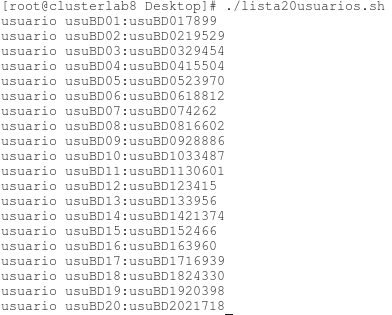




**Ejercicio 4:** **Realice un script que añade usuarios al cluster rocks y devuelve como salida el nombre y el password de cada usuario. El nombre debe ser usuBDxx, donde xx se incrementa de 01 a 20 y utilice una política de passwords razonable.**

**¿Cómo puede verificar cuántos usuarios existen en el cluster y que nombre tienen?**

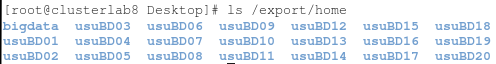
Al ejecutar el script la salida es la siguiente:



Con el comando “ls /export/home” se nos muestran los usuarios que se encuentran creados y con “ls /export/home | wc -w” te muestra el número total de usuarios. A continuación se mostrarán dos imágenes con la salida de cada uno de los dos comandos mencionados.

* Salida del comando **“ls /export/home | wc -w”**:



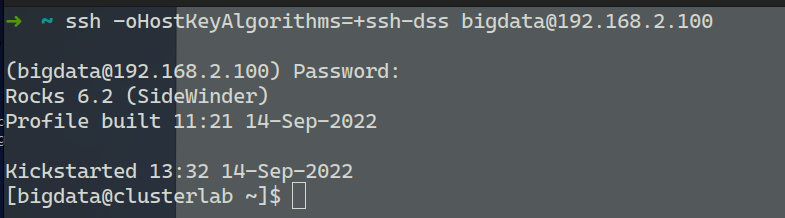
* Salida del comando **“ls /export/home”**:

Se puede ver que en los 20 usuarios se han creado correctamente y que en total muestra 21 usuarios porque cuenta el usuario “bigdata” que se creó anteriormente.

**Ejercicio 5:** **Realice las siguientes pruebas de conexión entre el las MV.**

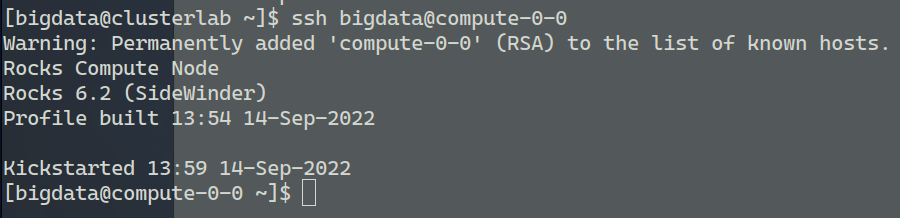
**5.1.- Conexión desde el anfitrión al cluster ( sería el equivalente a un acceso en remoto);**

**$ ssh** [**bigdata@192.168.182.100**](mailto:bigdata@192.168.182.100)



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

$ ssh compute-0-0



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

$ ssh [bigdata@10.11.12.](about:blank)252

Si intentamos conectar desde el anfitrión a uno de los nodos fallará, es decir, no establecerá ninguna conexión, esto se debe a que cuando configuramos el cluster se añadieron dos adaptadores de red, uno el eth1 y otro el eth0, siendo el eth1 el que permite al frontend conectarse con lo que llamamos "Public Ethernet" y con eth0 nos permite conectarnos a "Ethernet Network" donde el frontend permite conectar con el resto de nodos de la red, es por eso por lo que el anfitrión no se puede conectar directamente con un nodo de cómputo.

**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**

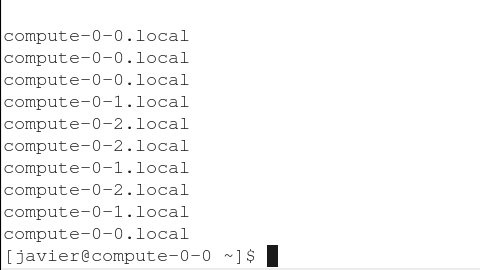
**Pruebe el siguiente comando:**

**$ /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**

Para la ejecución de nodos en paralelo, tras crear el txt de ejemplo donde se especifican los nodos de cómputo que van a ejecutar de manera paralela el comando “*hostname*”, un número total de 10 veces:



Como se puede apreciar, cada máquina realiza de manera concurrente el comando, y en nuestro caso, 4 veces el nodo de cómputo 0; 3 veces el nodo 1, y otras 3 veces el nodo 2; siendo el total de 10 veces que pedíamos.

Mientras que si ejecutamos tan solo el segundo comando, solo va a poner el host donde se esté ejecutando el comando, ya que no hemos especificado ningún nodo distinto.

Responda a las siguientes preguntas:

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

No se han ejecutado procesos en el frontend ya que lo hemos configurado para que solo sean los nodos de cómputo, los que realicen la operación.

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

Se modifica el fichero txt añadiendo a esta lista de nodos, el hostname del nodo frontend, siendo este “*clusterlab.uam.es*”.



Como se puede apreciar ahora, el nodo frontend actúa esta vez en la operación.

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

Habiendo ejecutado el comando desde cada máquina conectando mediante ssh, y variando el número de ejecuciones (para ver si cambiaba algo), se puede apreciar que dependiendo de la sesión en la que me encuentre (ej: compute-0-0), las primeras ejecuciones se van a realizar en ese propio nodo, lo cual tiene sentido porque al fin y al cabo, es el que recibe la orden más rápido.

Sobre el reparto de tareas, no tiene pinta de que haya ningún balanceador de carga, porque no se ha podido encontrar un patrón, con lo cual creemos que hay condiciones de carrera por ejecutar el comando.

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



En la ejecución de test, se puede ver como lo primero que hace el clúster es asignar procesos de ejecución a todos los nodos del clúster. A continuación, se comunica con ellos, mandando mensajes, y recibiendo la correspondiente respuesta.

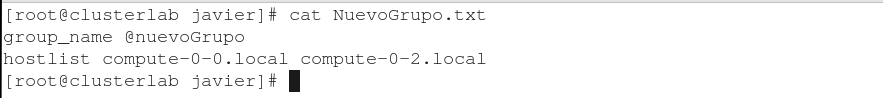
**Ejercicio 8:** **Crear una cola denominada colapares.q basándose en la cola all.q que tenga solo los nodos con nombres compute-0-x, siendo x par. Compruebe su funcionamiento.**

Lo primero que debemos hacer es crear un nuevo grupo de hosts que tan solo contenga los nodos de cómputo pares. Para ello:

1. Accedementos al perfil de superusuario con “*su*”
2. Volcamos el contenido de la ejecución del comando que contiene todos los hosts a un fichero, para así poder modificarlo:

“*qconf -shgrp @allhosts > ./NuevoGrupo.txt*”

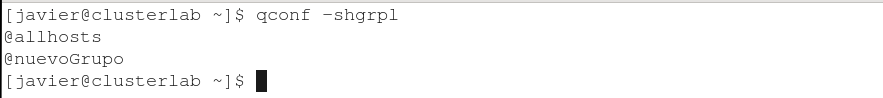
1. A continuación, modificamos el fichero, cambiamos el nombre del grupo de hosts, además, de eliminar el la lista el nodo de cómputo 1:



1. Ejecutamos el comando para crear un nuevo grupo de hosts a través del fichero recientemente modificado:

“*qconf -Ahgrp ./NuevoGrupo.txt*”

1. Finalmente, comprobamos que el nuevo grupo ha sido creado:

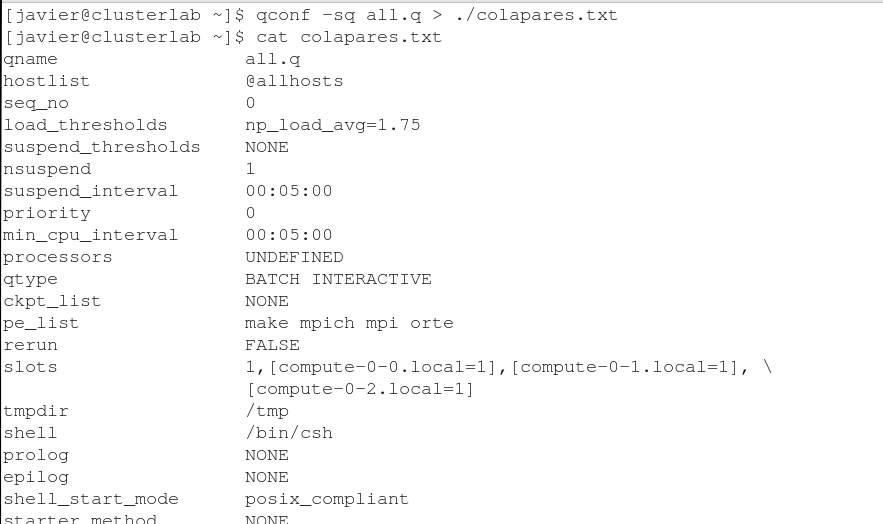


Una vez que hemos creado el nuevo grupo, podremos ya crear el script que, a diferencia del all.q, que contiene todos los hosts @allhosts, tenga nuestro subconjunto de @nuevoGrupo:

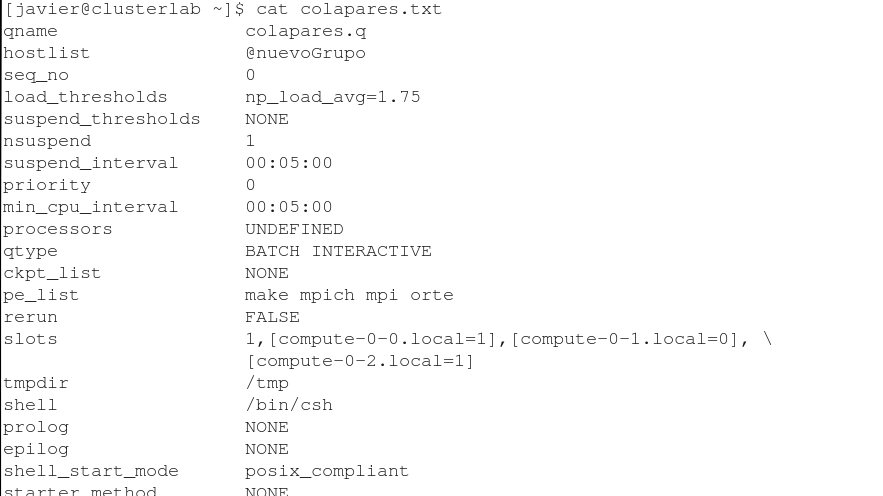
1. Accedemos a ver todas las colas con “*qconf -sql*”



1. Como hemos mencionado, solo tenemos la de all, pero a continuación, volcamos la configuración de esa cola a otro fichero, que llamamos “*colapares.txt*”:



1. Como podemos observar, en el hostlist, tenemos @allhosts, asignado, y ese el el campo, además del nombre, y de los “slots”, que deberemos modificar, quedando de la siguiente manera:



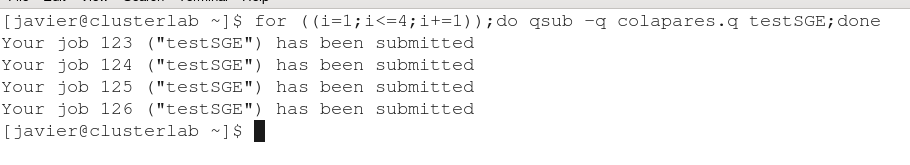
1. Por último, volcamos el contenido a las colas del gestor SGE con el siguiente comando, siendo superusuario:

“*qconf -Aq ./colapares.txt*”

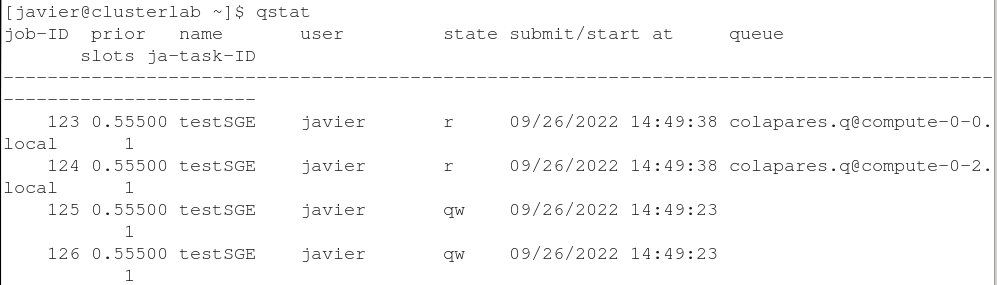
1. Comprobamos que se ha añadido:



Ahora la cola ya está lista para ser utilizada, para comprobarlo, vamos a usar el fichero “testSGE”, que ya utilizamos con anterioridad, y en un bucle de 2 iteraciones vamos a ejecutarlo, pero importante, añadiendo el parámetro “*-q colapares.q*”, para especificar que queremos tan solo esa cola, si no cogerá tanto “all.q”, como “colapares.q”:

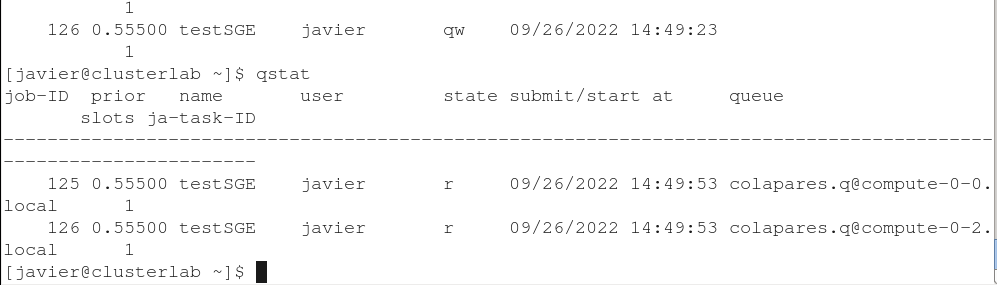


Ahora comprobamos con qstat:



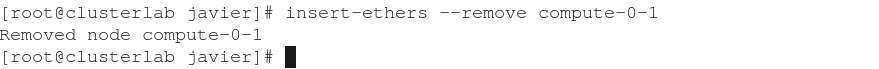
Se puede observar que ha cogido solo dos tareas con los nodos de cómputo 0 y 2 (pares), y además nos especifica que vienen de “*colapares.q*”.

Finalmente, esperamos a que se ejecute y observamos las dos últimas tareas:



**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.**

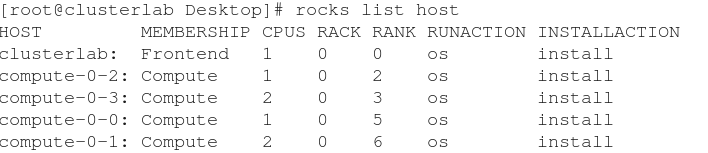
Lo primero que realizamos es forzar la desinstalación del nodo de cómputo 1, para volver a instalarlo con 2 cores:



Una vez configuramos la máquina con dos cores, desde el terminal de sudo, esperamos a insertar el nuevo nodo de cómputo con el mismo nombre con el siguiente comando:



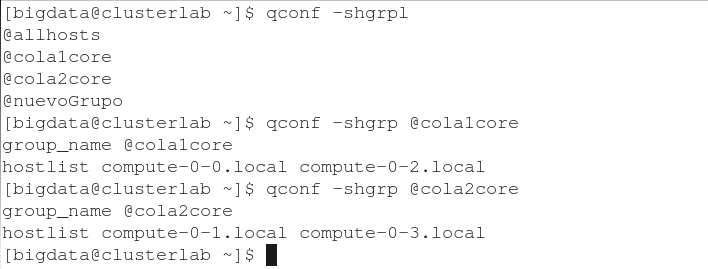
A continuación una vez instalado, observamos los nodos, procederemos a crear otro nuevo de dos cores (nodo de cómputo 3), y lo insertamos con el mismo comando, solo que no es necesario añadirle nombre personalizado porque automáticamente lo va a llamar compute-0-3:



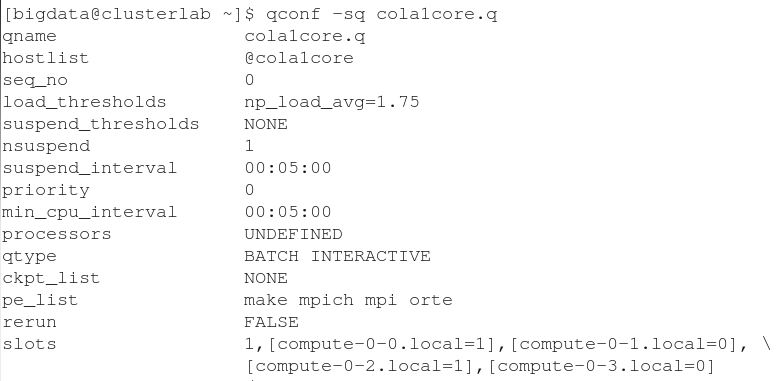
(Aparecen en ese orden porque tuve que borrar el 0, y hacer de nuevo el 1 por unos problemas)

La configuración restante es similar a la del ejercicio 8, tan solo cambian los nodos involucrados y los hosts añadidos a la lista.

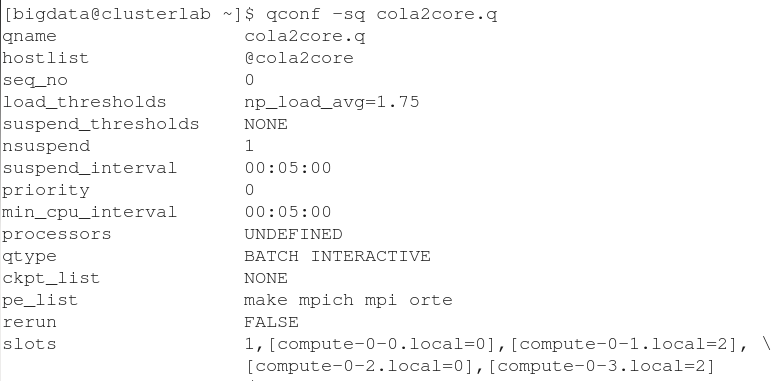
Los resultados en los hosts de los nodos de cómputo que usan un core y el otro de dos cores, son los siguientes:



Ahora vamos a ver como han quedado las colas, seleccionado los nodos involucrados:

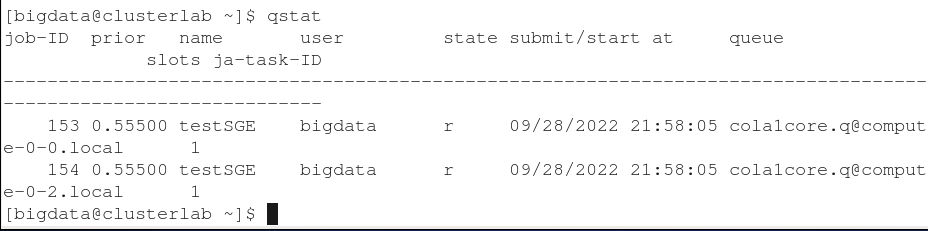
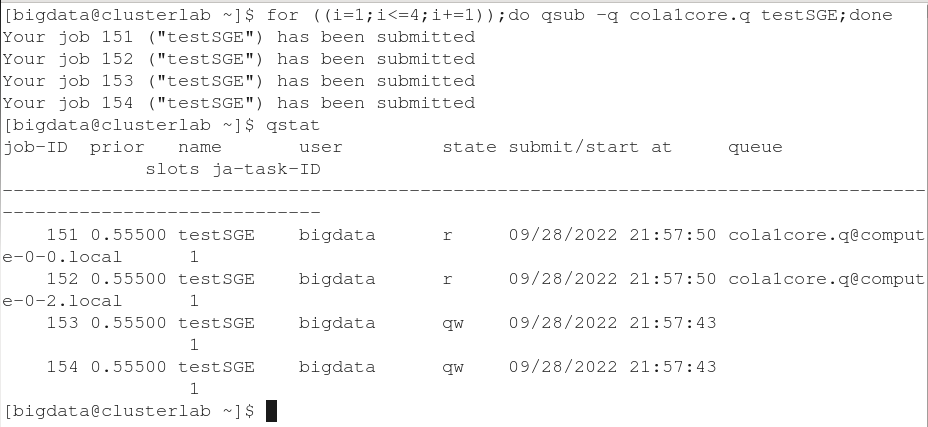


*cola1core.q*



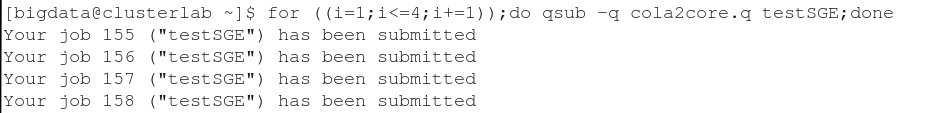
*cola2core.q*

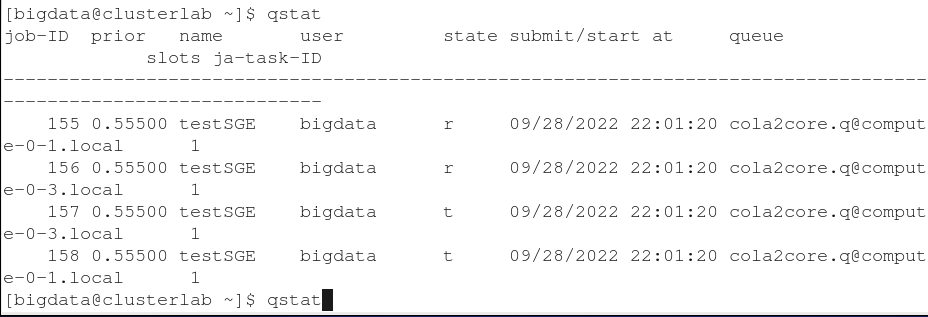
Ahora, vamos a comprobar para nodos de un core:



Donde vemos que la tarea la ejecuta los nodos 0 y 2, que son los de un core.

Ahora finalmente, vamos a comprobar para nodos de dos cores:





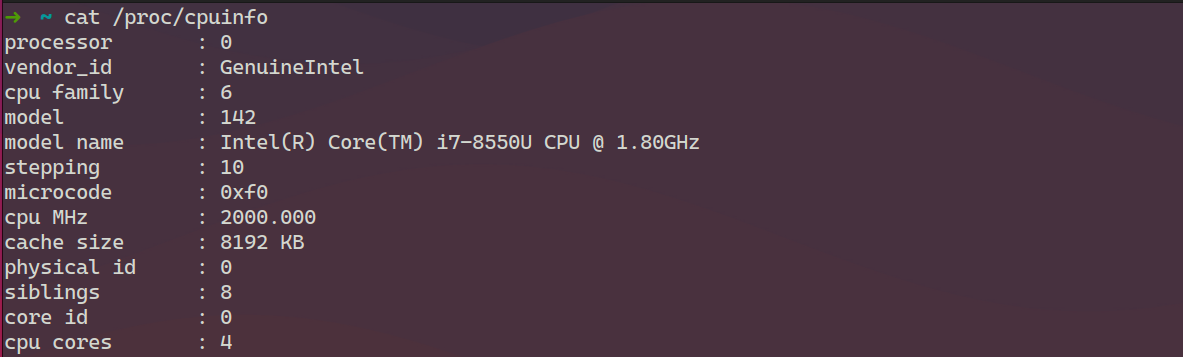
Como se puede ver, al tener dos cores cada nodo, se pueden ejecutar las cuatro tareas a la par, cada nodo cogiendo dos tareas.

**Práctica1 Parte 1 (35%)**

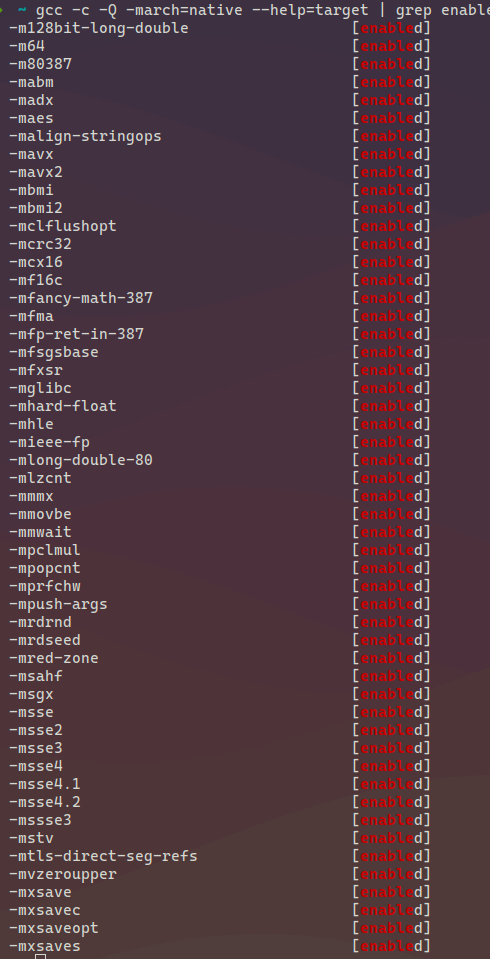
**Vector processing and SIMD**

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

Con el comando “cat /proc/cpuinfo” podemos apreciar cuál es el modelo de cpu de uno de los equipos personales de uno de los miembros de la pareja.



Con el comando “gcc -c -Q -march=native --help=target | grep enable” podemos ver cuales son las instrucciones SIMD soportadas por el procesador, siendo las instrucciones las siguientes:



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

Comparando los dos códigos ensambladores, el código vectorizado realiza más operaciones de ensamblador que el no vectorizado, pero la mayor diferencia se aprecia en que el vectorizado en la mayoría de sus operaciones son instrucciones de tipo “dp”, “pd” y “fd” mientras que el no vectorizado en su mayoría las instrucciones son de tipo “sd” y “ps”, lo que significa que el vectorizado realiza las operaciones de coma flotante con doble precisión mientras que el no vectorizado utiliza la precisión única. También el no vectorizado realiza operaciones con números escalares mientras que el vectorizado lo realiza con números empaquetados.

* **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.**

***for (t = 0; t < n; t++)***

***{***

***\_\_m256d mm = {1.0001, 1.0001, 1.0001, 1.0001};***

***\_\_m256d sum = {0.0, 0.0, 0.0, 0.0}; // to hold partial sums***

***for (i = 0; i < ARRAY\_SIZE; i += 4)***

***{***

***// Load arrays***

***\_\_m256d va = \_mm256\_load\_pd(&a[i]);***

***\_\_m256d vb = \_mm256\_load\_pd(&b[i]);***

***// Compute m\*a+b***

***\_\_m256d tmp = \_mm256\_fmadd\_pd(mm, va, vb);***

***// Accumulate results***

***sum = \_mm256\_add\_pd(tmp, sum);***

***}***

***// Get sum[2], sum[3]***

***\_\_m128d xmm = \_mm256\_extractf128\_pd(sum, 1);***

***// Extend to 256 bits: sum[2], sum[3], 0, 0***

***\_\_m256d ymm = \_mm256\_castpd128\_pd256(xmm);***

***// Perform sum[0]+sum[1], sum[2]+sum[3], sum[2]+sum[3], 0+0***

***sum = \_mm256\_hadd\_pd(sum, ymm);***

***// Perform sum[0]+sum[1]+sum[2]+sum[3]…***

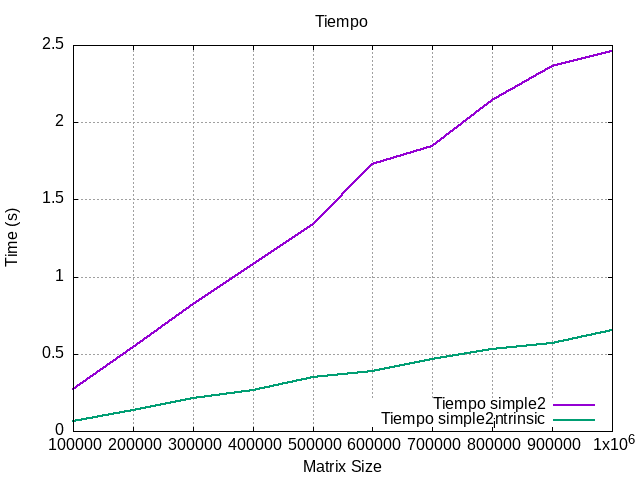
***sum = \_mm256\_hadd\_pd(sum, sum);***

***c += sum[0];***

***}***

Puesto que el objetivo es vectorizar el segundo bucle for, lo que se hace es guardar las suma parciales en el vector de sumas, esto es porque dicho *for* es el que se va a vectorizar y debemos incrementar el índice de 4 en 4 puesto que se van a operar 4 doubles en paralelo.

* + **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.**



Como se puede apreciar, la versión intrínseca tarda menos, puesto que la vectorización es el proceso por el cual aquellas operaciones matemáticas que se encuentren dentro de un bucle se ejecutan en paralelo, de modo que mientras que el simple realiza X iteraciones del segundo bucle, el intrínseco lo hace en X/4 veces, es decir, hace 4 veces menos de iteraciones ya que suma de 4 en 4 .

* **Exercise 3:**
  + **Compile and run the program using some images as arguments. Examine the results that were generated and analyze briefly the provided program.**

Tras realizar la ejecución del programa con unas cuantas imágenes, observamos que la imagen que se origina para cada una de las imágenes que se indican por argumentos pasa a estar en blanco y negro y pierde calidad de imagen frente a la original.

En lo que al código respecta, por cada imagen que recibe por parámetros (si es que puede abrirla) reserva memoria, a continuación para cada pixel de la imagen obtiene el valor de su escala de color RGB, es decir, el valor para el rojo (R), azul (B) y verde (G) y una vez que tiene dichos valores los multiplica por un cierto valor que depende si es para cambiar la cantidad de rojo, verde o azul y de esa forma cambiar el color del pixel a gris y una vez que ha modificado el color de todos los pixeles de la imagen, crea una nueva iamgen con dichos pixeles.

* + **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?**

Si, como se ha leído en el enunciado y se ha comentado anteriormente, la vectorización es el proceso por el cual aquellas operaciones matemáticas que se encuentran dentro de un bucle se realizan en paralelo. Puesto que el segundo bucle realiza operaciones de suma y multiplicación sobre los píxeles el término de vectorización podría aplicarse.

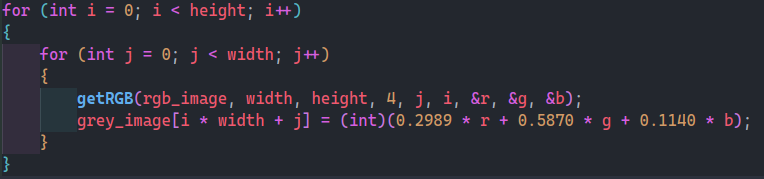
* + **Provide the source code of the auto-vectorized version of the code. Explain the changes in the code to help the compiler to vectorize the loop.**

Puesto que en la auto-vectorización el orden de acceso a los datos es importante debemos conseguir que la manera en la que en el bucle 1 (Loop 1) se recorre la memoria para obtener el valor de los píxeles sea más eficiente.

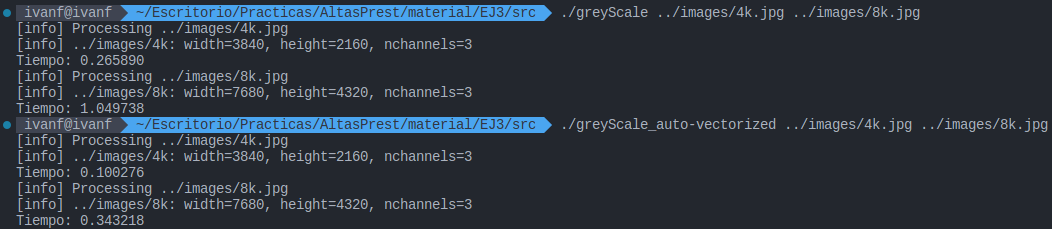
En el original las imágenes se crean de tal forma que se recorre antes el alto (height) o filas que el ancho (weight) o columnas y esta manera es menos eficiente a la hora de acceder a los datos de la caché.

Puesto que la memoria reservada para un vector es contigua en memoria será mejor acceder a los elementos en memoria siguiendo el orden en el que se encuentran en ella (por el principio de localidad).

A partir de lo comentado anteriormente nuestra solución consiste en recorrer antes el ancho, es decir, las filas que el alto o columnas, quedando ahora el bucle 1 de la siguiente manera:



Aunque en un apartado posterior se mostrará una tabla de tiempo, queremos adjuntar una imagen corroborando que con nuestra solución el tiempo se ha visto mejorado



Se puede apreciar una considerable mejora en los tiempos.

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

Partiendo del código auto-vectorizado (puesto que hay una mejora en lo que a tiempos respecta), los cambios realizados han sido los siguientes:

* + **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.**

A continuación se adjuntan dos tablas, la primera hará referencia a los tiempos obtenidos con la versión original y la segunda con la auto-vectorizada.

* **Original:**

| **Resolución** | **Tiempo** | **SpeedUp** | **Tasa de FPS** |
| --- | --- | --- | --- |
| **SD** | 0.003757 | 1 | 266.1698 |
| **HD** | 0.015163 | 1 | 65.95 |
| **FHD** | 0.031456 | 1 | 31.7904 |
| **4k** | 0.310426 | 1 | 3.2213 |
| **8k** | 1.088038 | 1 | 0.9191 |

* **Auto-vectorizada:**

| **Resolución** | **Tiempo** | **SpeedUp** | **Fps** |
| --- | --- | --- | --- |
| **SD** | 0.003799 | 1.01118 | 263.227 |
| **HD** | 0.013651 | 1.11076 | 73.2547 |
| **FHD** | 0.029591 | 1.06302 | 33.7940 |
| **4k** | 0.098414 | 3.15428 | 10.1612 |
| **8k** | 0.365293 | 2.97853 | 2.7375 |

* La aceleración se ha calculado de la siguiente forma: S = Toriginal / T
* La tasa de FPS se ha calculado de la siguiente manera: FPS = 1 / T
* **Discusión:**

Como se puede observar en cualquiera de las resoluciones el tiempo se ha visto mejorado, aunque para resoluciones bajas la mejora de tiempo no es sustancial, a medida que la resolución de la imagen aumenta se puede ver como la mejora es cada vez mayor.