UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Evaluación del uso de GPUs virtualizadas en sistemas de bajo consumo

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Palop Campos

Tutor: Federico Silla Jiménez

Carlos Reaño González

[2017/2018]

Resumen

El trabajo que a continuación se presenta, estudia el rendimiento y las prestaciones de ejecutar ciertas aplicaciones en distintos equipos, entre ellos uno de bajo consumo, utilizando su hardware local, así como su ejecución utilizando una GPU (Graphics Processing Unit) remota ubicada en la misma red local, mediante virtualización. Se quiere comprobar las diferencias de rendimiento dependiendo del hardware que contiene cada uno de los equipos elegidos.

Además, se quiere demostrar si la utilización de una GPU remota puede influir en el rendimiento y resultados de estas ejecuciones. Se utiliza un software llamado rCUDA, diseñado por el fabricante de GPU Nvidia, el cual es el encargado de poder usar una tarjeta gráfica de manera remota, ubicada en cualquier punto de una red, de esta manera poder aprovecharse de las prestaciones que ofrece esta tarjeta gráfica.

Palabras clave: virtualización, gpu, cuda, rCUDA, bajo consumo

Abstract

The work that is presented below, studies the performance and benefits of certain applications in different equipment, including one of low consumption, using its local hardware, as well as its execution using a remote GPU (Graphics Processing Unit) located in the same local network, through virtualization. You want to check the performance differences depending on the hardware that each of the chosen equipment contains.

In addition, we want to show if the use of a remote GPU can influence the performance and results of these executions. It uses a software called rCUDA, designed by the manufacturer of GPU Nvidia, which is responsible for being able to use a graphic card remotely, located at any point of a network, in this way to take advantage of the benefits offered this graphics card.

Keywords: virtualization, gpu, cuda, rCUDA

Resum

El treball que a continuació es presenta, estudia el rendiment y les prestacions de executar certes aplicacions en distints equips, entre ells un de baix consum, utilitzant el seu hardware local, aixi com la seua execució utilitzant una GPU (Graphics Processing Unit) remota ubicada en la mateixa xarxa local, a través de la virtualització. Es volen comprobar les diferencies de rendiment depenent dels components que contenen cada un dels equips de baix consum elegits.

A mes es vol demostrar si utilittzar una GPU remota pot influir en el rendiment y resultats d`aquestes execucions. Es utilitza un software anomenat rCUDA, disenyat per el fabricant de GPU Nvidia, el cual es el encarregat de poder utilizar una tarjeta gráfica de forma remota, ubicada en qualsevol lloc de una xarxa, d`aquesta manera poder aprofitar les prestacions que oferix aquesta tarjeta gráfica.

Paraules clau: virtualització, gpu, cuda, rCUDA, baix consum

Tabla de contenidos

Índice de imágenes y tablas

1. Introducción

1.1 Objetivos

El principal objetivo de este proyecto es estudiar el rendimiento de virtualizar una gpu mediante un software específico en un equipo de bajo consumo ubicado en algún punto de una red local. Además, se quiere comprobar que el rendimiento de virtualizar una GPU es similar a usar una GPU local.

La técnica de virtualización de GPU remota permite que una aplicación se ejecute en una computadora que no tenga instalado una GPU para hacer uso transparente del acelerador instalado en otro nodo del clúster.

1.2 Motivación

En el mundo tecnológico actual lo más importante es que un trabajo o una tarea se ejecuten en el mínimo tiempo posible, para aprovechar de una manera óptima los recursos de cualquier ámbito empresarial o laboral.

Actualmente la tecnología de la información está muy extendida y cada día avanza de manera exponencial, por lo que, en grandes clúster o centros de cálculo, el principal objetivo es que las ejecuciones de las distintas aplicaciones o el tiempo de computo, siempre sean en la mayoría de los casos lo más bajas posible. Esto es así, por la importancia de hoy en día de conseguir siempre los mejores resultados y prestaciones posibles que lleven a obtener unos mejores datos y llegar a ser líder o puntero en las investigaciones del mundo empresarial y tecnológico tan importante para las grandes empresas. Por lo que una tarjeta gráfica es indispensable para poder llevar a cabo las exigencias mencionadas anteriormente.

El problema que se plantea en este proyecto, es que en un clúster o centro de trabajo hay muchos nodos y de esta manera habría que poner a cada uno de estos una tarjeta gráfica, para llevar a cabo nuestros objetivos, pero esto acarrearía ciertas desventajas como, por ejemplo:

- Gran Costes de adquisición
- Consumo eléctrico (incremento de un 20–30%) incluida la refrigeración
- Mantenimiento (tecnológico y personal)
- Espacio
- No todas las aplicaciones pueden ser aceleradas por una GPU

En general una gran pérdida de recursos, y pocas ventajas generales, exceptuando las mayores prestaciones para el cómputo mediante las GPU. Debido a estas razones es interesante investigar sobre la virtualización de las gpus en clusters o centros de cálculo para comprobar que los resultados de ejecutar ciertos trabajos, estudios, o simulaciones son similares, independientemente si la gpu esta localmente en un nodo o esta misma se utiliza de manera remota.

El éxito de la virtualización conllevaría transformar los problemas anteriormente mencionados en ventajas.

1.3 Conexión a clusterTFG.upv.es y Nodos

Para mi trabajo y ejecuciones he usado un escenario real, concretamente en el clusterTFG.upv.es de la Universidad Politécnica de Valencia (UPV).

Se han usado los nodos que la universidad me ha prestado, para realizar las distintas pruebas que necesitaba para poder desarrollar mi TFG (Trabajo de fin de grado).

El procedimiento de conexión a los distintos nodos ha sido el siguiente:

Mediante Putty me he conectado a clusterTFG.upv.es y al nodo compartido f1.

A la hora de transferir ficheros a mi ruta personal del nodo, he usado el programa WinScp, de esta manera los he subido de una manera sencilla.

He subido todos los ficheros correspondientes de las suites a utilizar durante el desarrollo de mi trabajo

- Phoronix Test Suite
- Relion-Master
- Librerías Cuda
- Librerías rCUDA

Normalmente estos ficheros suelen ir comprimidos, para descomprimir se usa del comando tipo de descompresión de Linux: **tar –xzvf “nombrearchivo”.tar.gz**

En concreto los dos nodos que hemos utilizado durante el TFG para realizar las distintas pruebas y test han sido los siguientes:

nodo-arm – Nodo de bajo consumo con arquitectura Arm

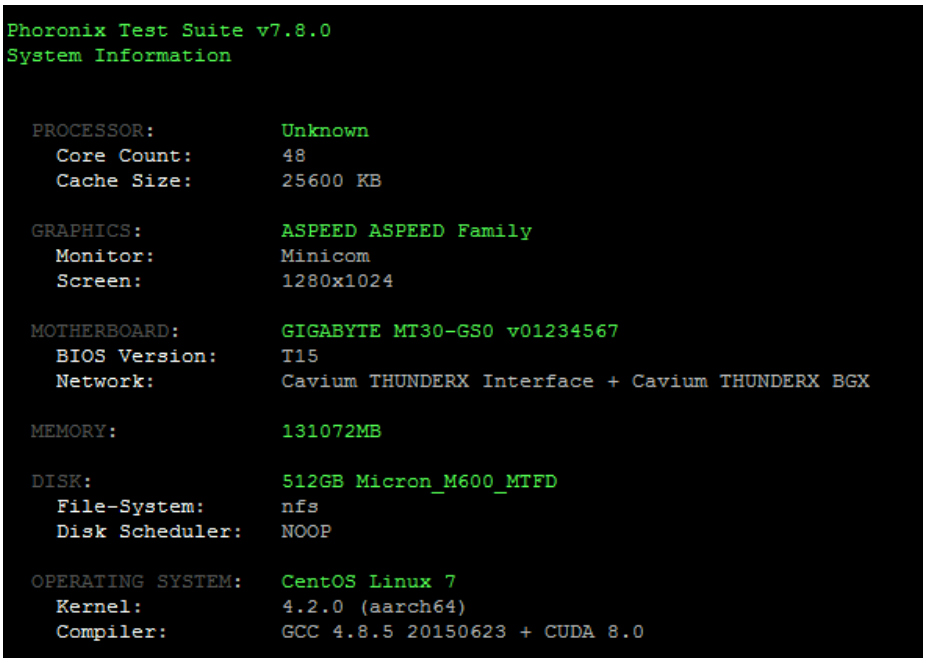

Figura 1.3.1 Especificaciones Nodo nodo-arm

nodo-x86 – Nodo con Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz

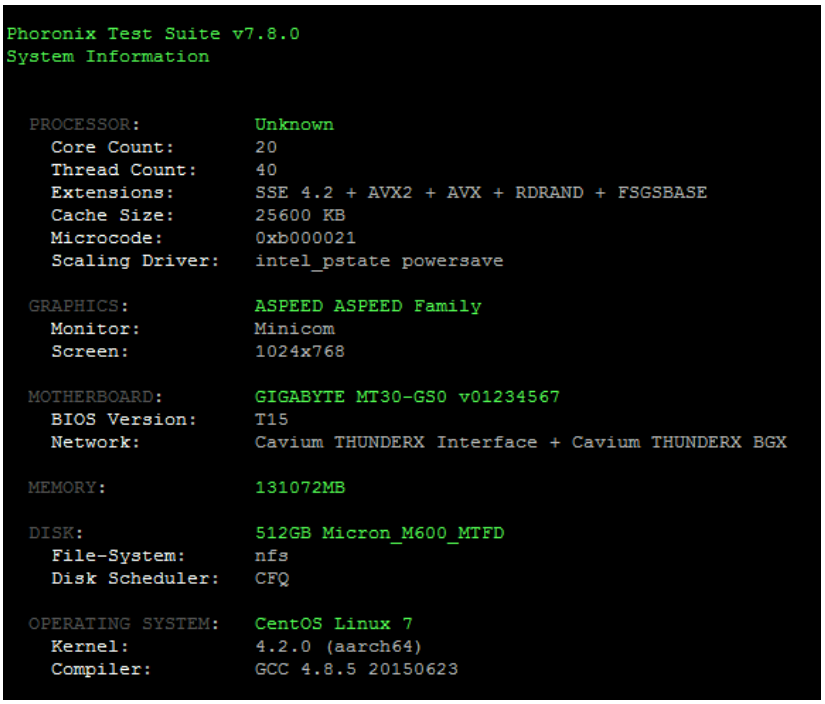

Figura 1.3.2 Especificaciones Nodo nodo-arm

2. Estado del Arte

En este apartado se habla de las principales características de algunos de los componentes tecnológicos actuales, CPU, arquitecturas de bajo consumo, tarjetas gráficas, tecnologías de red, etc. Mostraremos las principales ventajas y características de cada una de ellas y si existen algunas diferencias significativas entre estas tecnologías.

2.1 Cpu`s: Xeon, Atom, ARM

En este punto se dan a conocer dos de las principales tecnologías actuales y más comunes de CPU, centrándonos además en la arquitectura ARM, la cual es una arquitectura de bajo consumo.

Arquitectura ARM

- Está basada en una arquitectura RISC (Reduced Instruction Set Computer=Ordenador con Conjunto Reducido de Instrucciones) de 32 bits.

- A partir de la llegada de su versión V8-A, también se incluyó una arquitectura de 64 Bits.

- La arquitectura ARM es el conjunto de instrucciones de 32 y 64 bits más ampliamente utilizado en unidades producidas actualmente por los principales fabricantes.

- Este tipo de enfoque de diseño basado en RISC permite que los procesadores ARM requieran una cantidad menor de transistores que los procesadores x86 CISC, típicos en la mayoría de ordenadores personales.

- Estos tipos de procesadores contemplan una reducción de los costes, calor y energía.

- Estas características son deseables para dispositivos que funcionan con baterías, como los teléfonos móviles, tabletas, etc., debido a la razón arriba anteriormente mencionada.

- La relativa simplicidad con la que están fabricados los procesadores ARM los hace ideales para aplicaciones de baja potencia.

- Un dato importante es que, en 2005, alrededor del 98% de los más de mil millones de teléfonos móviles vendidos utilizaban al menos un procesador ARM, lo que muestra el impacto de este tipo de procesador en el mundo tecnológico.

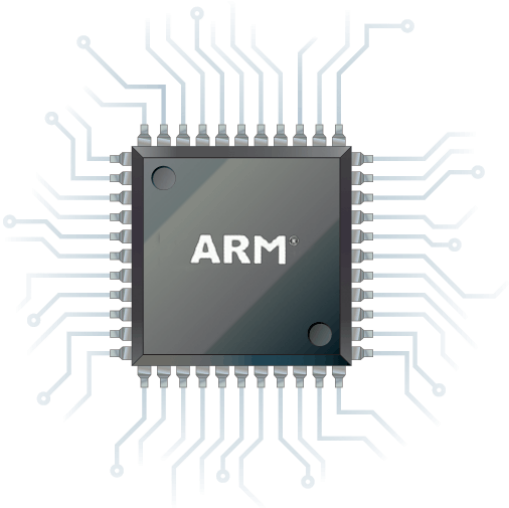

Figura 2.1.1 Representación procesador ARM

- Desde 2009, los procesadores ARM son aproximadamente el 90% de todos los procesadores RISC de 32 bits integrados.

- La fama de ARM es tal que en 2013 se ha convertido en la arquitectura de 32 bits más exitosa a lo largo y ancho del mundo si hablamos de cantidad de producción.

- Están licenciados por la compañía británica ARM Holdings, de esta manera la arquitectura ARM es licenciable, esto significa que licencia sus creaciones a otros fabricantes y esto ayuda a que se extienda su presencia en el mercado, así como los ingresos proporcionados.

- De esta manera los procesadores ARM pueden ser modificados por las empresas que compran su diseño.

- El negocio principal de ARM Holdings es la venta de núcleos IP, estas licencias se utilizan para crear microcontroladores y CPUs basados en este núcleo.

- El ARM2 es probablemente el procesador de 32 bits útil más simple del mundo, ya que posee sólo 30.000 transistores. Gracias a esto el consumo de energía es extremadamente bajo.

- Algunos de los nombres tan reconocidos como Tegra de Nvidia, OMAP de Texas Instruments o Snapdragon de Qualcomm, parten de la tecnología ARM y son muy habituales en los dispositivos móviles más famosos del sector.

- También el procesador A6 de Apple, se basan en la arquitectura ARM de una u otra manera.

- Los procesadores ARM están optimizados para realizar instrucciones mucho más sencillas, a muy bajo nivel, el procesador sólo tiene que seguir los pasos para conseguir que el código funcione.

- Debido a que estos procesadores consumen menos energía implica que emita menos calor, de esta manera estos procesadores no alcanzan temperaturas tan altas como nuestros procesadores de sobremesa, lo que hace que no sea necesario utilizar refrigeración tan potente en ellos.

Arquitectura ATOM

- Su uso está algo limitado debido a que están diseñados para mantener un bajísimo consumo y una potencia media.

- Son los procesadores de arquitectura x86 fabricados en 45 nanómetros con los transistores más pequeños del mundo.

- Tienen unos 47 millones de transistores en una superficie de 25 milímetros cuadrados.

- El nombre de Intel® Atom es el nombre de una línea de microprocesadores de ultra-baja tensión x86 y x86-64 de Intel.

- Originalmente diseñados para un proceso de fabricación de 45 nanómetros CMOS, los modelos posteriores a *Cedar* usan una tecnología de 32 nanómetros.

- Los Atom implementan el conjunto de instrucciones x86-64 y x86 (IA-32).

Figura 2.1.2 Representación procesador Intel Atom

Arquitectura Xeon

- Xeon es una familia de microprocesadores Intel para servidores PC y Macintosh. El primer procesador Xeon apareció en 1998 con el nombre de Pentium II Xeon.

- Actualmente también es usado por muchos servidores que ofrecen hostings en internet, dado a su rendimiento y velocidad de los modernos y actuales procesadores Xeon.

- El 26 de junio de 2006, Intel anunció la nueva generación Xeon Dual Core con tecnología de doble núcleo. Intel afirma que este nuevo procesador brinda un 80% más de rendimiento por vatio y es un 60% más rápido que la competencia AMD. Además la nueva generación ofrece más del doble de rendimiento que la generación anterior de servidores basados en el procesador Intel Xeon; es capaz de ejecutar aplicaciones de 32 y 64 bits.

- Igualmente, este último procesador sustituyó al veterano PowerPC en las estaciones de trabajo MacPro y también su nuevo modelo del año 2013 y los servidores XServe de Apple cuando se hizo la transición de Power PC a x86, mejorando su eficacia con la tecnología de arranque EFI.

2.2 Graphics Processing Units: CUDA, K20, K40, P100

En este apartado vamos a comentar algunas de las principales GPU Nvidia que hay en la actualidad, mostrando sus principales ventajas y características.

CUDA

- Es una arquitectura de cálculo paralelo de nVidia, incluye un compilador y un conjunto de herramientas de desarrollo creadas por nVidia que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de Nvidia..

- Aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

- Gracias a millones de GPUs CUDA vendidas hasta la fecha, miles de desarrolladores, científicos e investigadores están encontrando innumerables aplicaciones prácticas para esta tecnología en una gran diversidad de campos.

- Por medio de wrappers (son clases diseñadas para ser un complemento de los tipos primitivos) se puede usar Python, Fortran y Java en vez de C/C++.

- Funciona en todas las GPU nVidia de la serie G8X en adelante, incluyendo GeForce, Quadro, ION y la línea Tesla. nVidia afirma que los programas desarrollados para la serie GeForce 8 también funcionarán sin modificaciones en todas las futuras tarjetas nVidia, gracias a la compatibilidad binaria.

Nvidia Tesla K20

- Utiliza la GPU Nvidia GK110, cuya principal característica es la de tener 7.1 mil millones de transistores, una cifra abrumadora que duplica lo que solemos encontrar en los chips gráficos domésticos más potentes del fabricante.

- Año 2012.

- 28 nanómetros.

- 1.5 TFLOPS en punto flotante de doble precisión.

- Distintas versiones con 6, 12 y 24 GB de memoria Ram, todas con interfaz de 384 bits.

- La familia Tesla K20 acelera la mayor variedad de aplicaciones comerciales, de ciencia e ingeniería que se utilizan en los CPD y en los centros de alta computación.

Figura 2.2.1 Tarjeta Gráfica Nvidia Tesla K20

Nvidia Tesla K40

Figura 2.2.2 Tarjeta Gráfica Nvidia Tesla K40

- Año 2013.

- 12 GB GDDR5 de memoria Ram.

- Está diseñada para aplicaciones de rendimiento extremo en los campos de la aplicación empresarial.

- Para tareas de trabajo pesado cuenta con el doble de memoria que su predecesor (Tesla K20x) y hasta un 40 por ciento más de rendimiento.

- Ofrece 4,29 teraflops de precisión simple y 1.43 teraflops cuando hablamos del rendimiento de la coma flotante de doble precisión.

- Hay más de 240 aplicaciones de software profesionales que aprovechan la aceleración de esta GPU.

Tesla K20 vs Tesla K40

GPUS TESLA PARA ESTACIONES DE TRABAJO		
Características fundamentales	Tesla K40	Tesla K20
Pico de rendimiento de operaciones en coma flotante de doble precisión	1.43 Tflops	1.17 Tflops
Pico de rendimiento de operaciones en coma flotante de precisión simple	4.29 Tflops	3.52 Tflops
Ancho de banda de memoria (ECC desactivada)	288 GBytes/s	208 GBytes/s
Cantidad de memoria (GDDR5)	12 GB	5 GB
Núcleos CUDA	2880	2496

Figura 2.2.3 Tesla K40 vs Tesla K20

Nvidia Tesla P100

Figura 2.2.4 Tarjeta Gráfica Nvidia Tesla P100

-Año 2017.

-Interfaz PCI-Express/NVLink.

-Memoria apilada HBM2.

- Orientada a los servidores.

-12 o 16 GB de memoria HBM2.

- GPU Pascal GP102.

- 3840 Cuda Cores y 240 TMUs.

- Permite alcanzar un ancho de banda máximo de **720 GB/s.**

- 9,3 TFLOPs en precisión simple y 4.7 TFLOPs en precisión doble en el modelo PCI-Express.

- 10.6 TFLOPs y 5.3 TFLOPs en la variante NVLink.

	P100 para servidores PCIe	P100 para servidores optimizados con NVLink
Rendimiento con precisión doble	4,7 TeraFLOPS	5,3 TeraFLOPS
Rendimiento con precisión simple	9,3 TeraFLOPS	10,6 TeraFLOPS
Rendimiento con precisión media	18,7 TeraFLOPS	21,2 TeraFLOPS
Ancho de banda del enlace NVIDIA NVLink™	-	160 GB/s
Ancho de banda del enlace PCIe x16	32 GB/s	32 GB/s
Pilas de memoria HBM2 sobre CoWoS	16 GB o 12 GB	16 GB
Ancho de banda de las pilas de memoria HBM2 sobre CoWoS	732 GB/s o 549 GB/s	732 GB/s
Programabilidad mejorada con motor de migración de páginas	✓	✓
Protección ECC para asegurar la fiabilidad de los datos	✓	✓
Optimizada para servidores de centros de datos	✓	✓

Figura 2.2.5 Tabla características interfaces Nvidia Tesla P100

2.3 Nvidia Cuda y rCUDA

Llegados a este punto vamos a explicar cómo hemos instalado y ejecutado el software CUDA en nuestros nodos, y los valores obtenidos por dichos test.

Lo mismo vamos a realizar con el software rCUDA (remote CUDA), donde desde un nodo cliente utilizaremos la gpu de un nodo que actuara como servidor.

Para rCUDA probaremos los test con la tecnología de red GIGABIT

Cuda

Primero que nada, nos hemos familiarizado con la tecnología CUDA en el nodo compartido f1 para poder lanzar los test que hemos decidido utilizar.

Lo primero que hemos realizado ha sido compilar CUDA, para más tarde ejecutarlo:

Para poder compilar CUDA tendremos que importar a nuestro PATH la librería correspondiente a CUDA ubicada en los distintos directorios dependiendo de la arquitectura del sistema a ejecutar:

/lib/Intel/CUDA/8.0/lib64 (Arquitectura x86)

/lib/arm/CUDA/8.0/bin (Arquitectura ARM)

Se ha usado el siguiente comando para exportar esta librería a nuestro PATH:

nodo-x86 (x86)

export LD_LIBRARY_PATH=/lib/intel/CUDA/8.0/lib64:$LD_LIBRARY_PATH

cd /lib/intel/CUDA/8.0/bin

nodo-arm (Arm)

export LD_LIBRARY_PATH=/lib/arm/CUDA/8.0/lib64:$LD_LIBRARY_PATH

cd /lib/arm/CUDA/8.0/bin

Una vez ejecutado este comando, ya tenemos lo necesario para poder ejecutar los test correspondientes de CUDA, en concreto vamos a pasar los test:

-deviceQuery: Muestra la información de nuestra GPU.

-bandwitdhTest: es una utilidad para medir el ancho de banda de la memoria entre la CPU y la GPU y entre direcciones en la GPU.

-matrixMul: Resultado de cómputo usando el kernel Nvidia.

Ejecución Test Cuda en nodo nodo-x86

-Test deviceQuery

La ruta correspondiente a este test se encuentra en nuestra carpeta de cuda, exactamente en la ruta:

/home/alumno/NVIDIA_CUDA_Samples/x86_64/8.0/1_Utilities/deviceQuery

Una vez situados en esta carpeta, se compilará el test para que podamos ejecutarlo. Se ha compilado con el siguiente comando:

make EXTRA_NVCCFLAGS=--cudart=shared

Y ejecutamos con el comando: ***./deviceQuery***

```
[japacam@mlxc18 deviceQuery]$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla P100-PCIE-16GB"
  CUDA Driver Version / Runtime Version          9.1 / 8.0
  CUDA Capability Major/Minor version number:    6.0
  Total amount of global memory:                 16281 MBytes (17071734784 bytes
)
  (56) Multiprocessors, ( 64) CUDA Cores/MP:     3584 CUDA Cores
  GPU Max Clock rate:                            1329 MHz (1.33 GHz)
  Memory Clock rate:                             715 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536)
, 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 130 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simu
ltaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.1, CUDA Runtime Versi
on = 8.0, NumDevs = 1, Device0 = Tesla P100-PCIE-16GB
Result = PASS
```

Figura 3.2.1 Test deviceQuery en nodo-x86

-Test bandwidthTest

La ruta correspondiente a este test se encuentra en nuestra carpeta de cuda , exactamente en la ruta:

/home/alumno/NVIDIA_CUDA_Samples/x86_64/8.0/1_Utilities/bandwidthTest

Una vez situados en esta carpeta, se compilará el test para que podamos ejecutarlo. Se ha compilado con el siguiente comando:

make EXTRA_NVCCFLAGS=--cudart=shared

Y ejecutamos con el comando: ***./bandwidthTest***

```
[japacam@mlxc18 bandwidthTest]$ ls
bandwidthTest  bandwidthTest.cu  bandwidthTest.o  bandwidthTest_static  Makefile  NsightEclipse.xml  readme.txt
[japacam@mlxc18 bandwidthTest]$ ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: Tesla P100-PCIE-16GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     11843.1

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     12858.8

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     499738.9

Result = PASS
```

Figura 3.2.2 Test bandwidth en nodo-x86

-Test matrixMul

La ruta correspondiente a este test se encuentra en nuestra carpeta de cuda , exactamente en la ruta:

/home/alumno/NVIDIA_CUDA_Samples/x86_64/8.0/0_Simple/matrixMul

Una vez situados en esta carpeta, se compilará el test para que podamos ejecutarlo. Se ha compilado con el siguiente comando:

make EXTRA_NVCCFLAGS=--cudart=shared

Y ejecutamos con el comando: **./matrixMul**

```
[japacam@mlxc18 matrixMul]$ ./matrixMul
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Tesla P100-PCIE-16GB" with compute capability 6.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 1730.05 GFlop/s, Time= 0.076 msec, Size= 131072000 Ops, WorkgroupSize= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

Figura 3.2.3 Test matrixMul en nodo-x86

rCUDA

El software rCUDA se estructura siguiendo una arquitectura distribuida cliente-servidor, donde el cliente se ejecuta en el mismo nodo que está demandando la utilización de una GPU, mientras que el servidor es ejecutado en el nodo donde físicamente se encuentra instalado uno de estos aceleradores.

Además, permite el uso de distintas tecnologías de comunicación gracias a su arquitectura modular, que soporta la elección de bibliotecas de comunicación en tiempo de ejecución. rCUDA actualmente soporta módulos de comunicación sobre Ethernet e InfiniBand aprovechando el incremento de prestaciones obtenido en la última tecnología FDR de InifiniBand.

Para rCUDA, tenemos que disponer de un cliente y un servidor, en nuestro caso los nodos nodo-arm y nodo-x86 respectivamente. El cliente usara la gpu del servidor de manera remota mediante rCUDA. De esta manera el nodo-arm de bajo consumo, usara la gpu del nodo-x86 para la ejecución de los test de Cuda

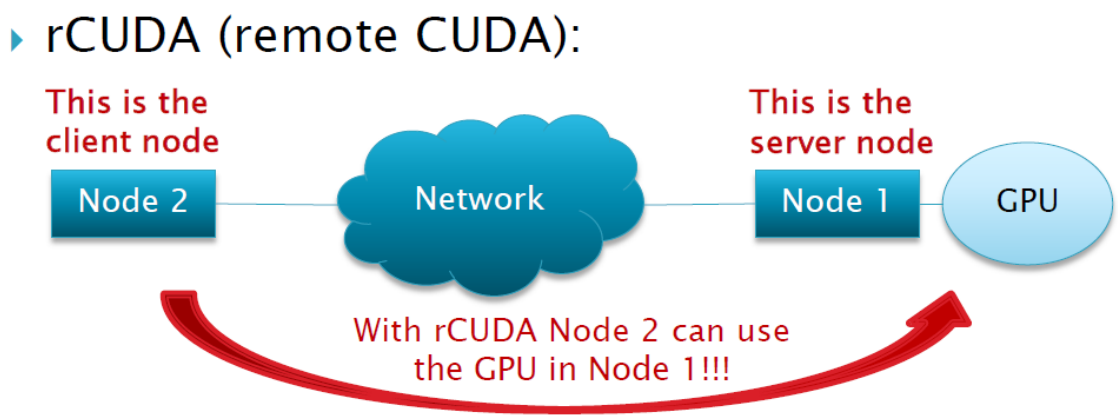

Figura 3.2.4 Esquema resumen rCUDA

Características del **cliente:**

- Emula las GPU locales

- Maneja la comunicación con el servidor

Características del **servidor:**

- Espera solicitudes

- Ejecuta código de GPU (kernel)

- Informes de estado

-Ejecución en el servidor nodo-x86 de la aplicación rCUDA.

De nuevo añadiremos en nuestro nodo las librerías de rCUDA comprimidas a nuestro directorio correspondiente. Las descomprimimos con el comando tar -xzvf archivo.tar.gz.

Como en el caso de CUDA, debemos suministrar a nuestro nodo las librerías correspondientes para que podamos usar rCUDA. De esta manera haremos un export de las librerías para que estén contenidas en nuestro path.

Hemos ejecutado los siguientes comandos en el servidor nodo-x86 para ejecutar rCUDA

1 Paso:

export LD_LIBRARY_PATH=/lib/intel/CUDA/8.0/lib64:*/lib/intel* /LIBS/CUDNN/8.0-v6.0/lib64

cd /home/alumno/tfg/rCUDAv18.03beta-CUDA8.0-cuDNN6.0/x86_64/bin

2 Paso:

./rCUDAd –iv (con estas opciones podremos ver como se conecta el cliente)

Para usar la interfaz infiniband usamos el comando:

export RCUDAPROTO=IB

-Ejecución en cliente nodo-arm de CUDA

1 Paso: (exportamos librería de cuda y rCUDA a nuestro cliente)

export LD_LIBRARY_PATH=/home/alumno/tfg/rCUDAv16.11.04.02-CUDA8.0/lib:/home/alumno/tfg/rCUDAv18.03beta-CUDA8.0-cuDNN6.0/aarch64/lib

2 Paso: (le decimos al cliente que tiene que usar una gpu remota)

export RCUDA_DEVICE_COUNT=1

3 Paso: (con este comando le decimos al cliente cual es el servidor)

export RCUDA_DEVICE_0=nodo-x86

4 Paso: (ejecución del test CUDA correspondiente)

/home/alumno/NVIDIA_CUDA_Samples/aarch64/8.0/1_Utilities/bandwidthTest

./deviceQuery

Para usar la interfaz infiniband usamos el comando:

export RCUDAPROTO=IB

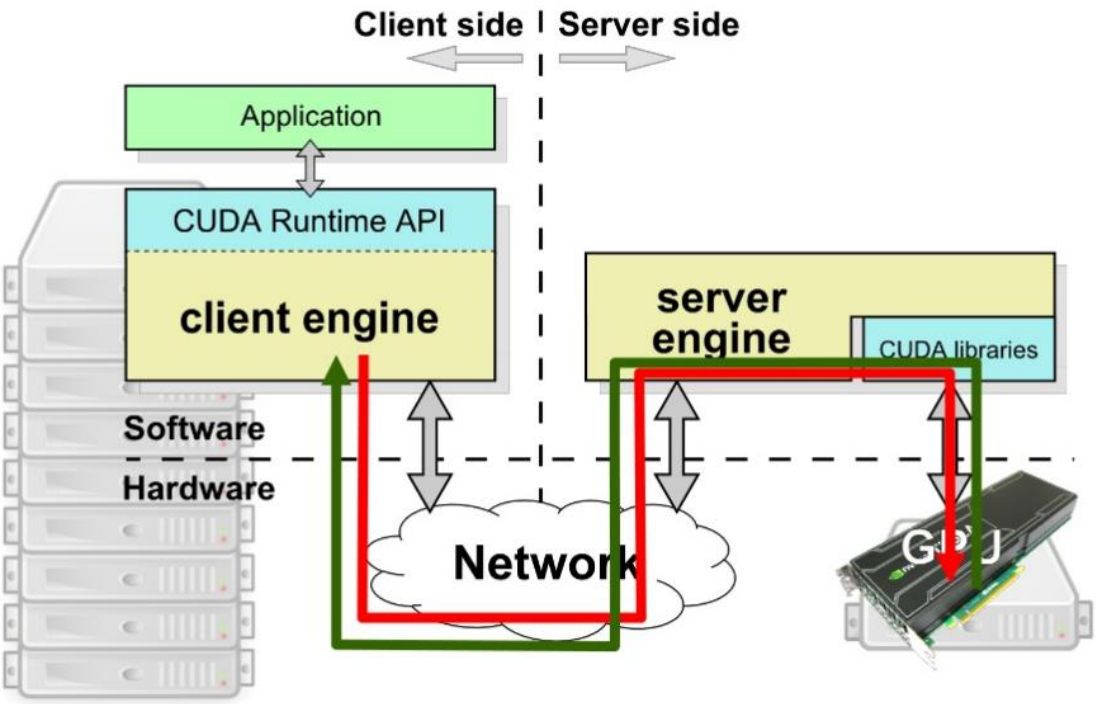

Figura 3.2.5 Esquema resumen rCUDA

• El software cliente recibe una petición CUDA de la aplicación.

• La petición es procesada y enviada al servidor.

• El servidor en el nodo remoto interpreta y ejecuta la petición sobre una GPU real.

• Cuando la GPU finaliza la ejecución, el servidor devuelve el resultado al cliente.

• El cliente devuelve el resultado a la aplicación en ejecución

Ejecución Test Cuda en nodo nodo-arm con rCUDA

-Test deviceQuery

La ruta correspondiente a este test se encuentra en otra ruta, ya que al ser un nodo de arquitectura ARM, las librerías están ubicadas en otra carpeta de cuda, exactamente en la ruta:

/home/alumno/NVIDIA_CUDA_Samples/aarch64/8.0/1_Utilities/deviceQuery

Una vez situados en esta carpeta, se compilará el test para que podamos ejecutarlo. Se ha compilado con el siguiente comando:

make EXTRA_NVCCFLAGS=--cudart=shared

Y ejecutamos con el comando: ***./deviceQuery***

```
[japacam@mlxarm1 deviceQuery]$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla P100-PCIE-16GB"
  CUDA Driver Version / Runtime Version          9.1 / 8.0
  CUDA Capability Major/Minor version number:    6.0
  Total amount of global memory:                 16281 MBytes (17071734784 bytes)
  (56) Multiprocessors, ( 64) CUDA Cores/MP:     3584 CUDA Cores
  GPU Max Clock rate:                            1329 MHz (1.33 GHz)
  Memory Clock rate:                             715 Mhz
  Memory Bus Width:                              4096-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 130 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.1, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = Tesla P100-PCIE-16GB
Result = PASS
```

Figura 3.2.6 Test deviceQuery en nodo-arm

-Test bandwidthTest

La ruta correspondiente a este test se encuentra en nuestra carpeta de cuda, exactamente en la ruta:

/home/alumno/NVIDIA_CUDA_Samples/aarch64/8.0/1_Utilities/bandwidthTest

Una vez situados en esta carpeta, se compilará el test para que podamos ejecutarlo. Se ha compilado con el siguiente comando:

Hemos tenido que cambiar el makefile de este test para que ejecute las librerías de la arquitectura aarch64

make EXTRA_NVCCFLAGS=--cudart=shared

Y ejecutamos con el comando: ***./bandwidthTest –memory=pageable***

```
[japacam@mlxarm1 bandwidthTest]$ ./bandwidthTest --memory=pageable
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: Tesla P100-PCIE-16GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     114.8

 Device to Host Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     114.6

 Device to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     397520.9

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

Figura 3.2.7 Test bandwidthTest en nodo-arm

-Test matrixMul

La ruta correspondiente a este test se encuentra en nuestra carpeta de cuda, exactamente en la ruta:

/home/alumno/NVIDIA_CUDA_Samples/x86_64/8.0/0_Simple/matrixMul

Una vez situados en esta carpeta, se compilará el test para que podamos ejecutarlo. Se ha compilado con el siguiente comando:

Hemos tenido que cambiar el makefile de este test para que ejecute las librerías de la arquitectura aarch64

make EXTRA_NVCCFLAGS=--cudart=shared

Y ejecutamos con el comando: ./matrixMul

```
[japacam@mlxarm1 matrixMul]$ ./matrixMul
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "Tesla P100-PCIE-16GB" with compute capability 6.0

MatrixA(320,320), MatrixB(640,320)
Computing result using CUDA Kernel...
done
Performance= 1727.71 GFlop/s, Time= 0.076 msec, Size= 131072000 Ops, WorkgroupSiz
e= 1024 threads/block
Checking computed result for correctness: Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may va
ry when GPU Boost is enabled.
```

Figura 3.2.8 Test matrixMul en nodo-arm

Comparativa del Test matrixMul en los dos nodos del proyecto.

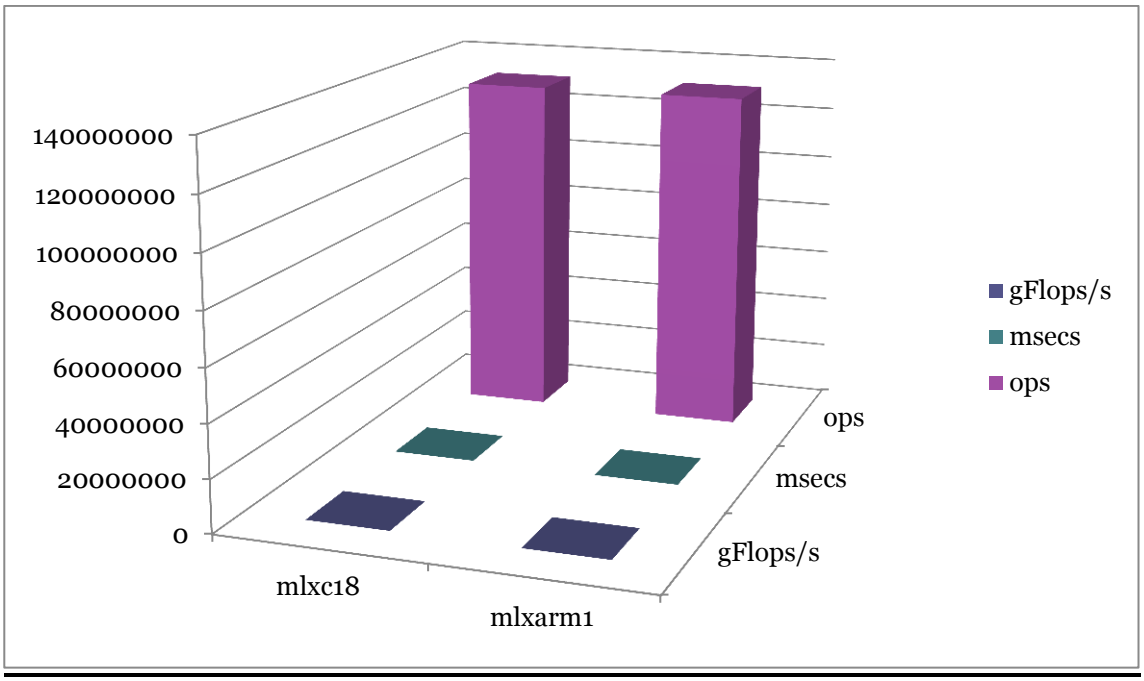

Figura 3.2.9 Comparativa test matrixMul en nodos proyecto

Comparativa del Test bandWidthTest en los dos nodos del proyecto.

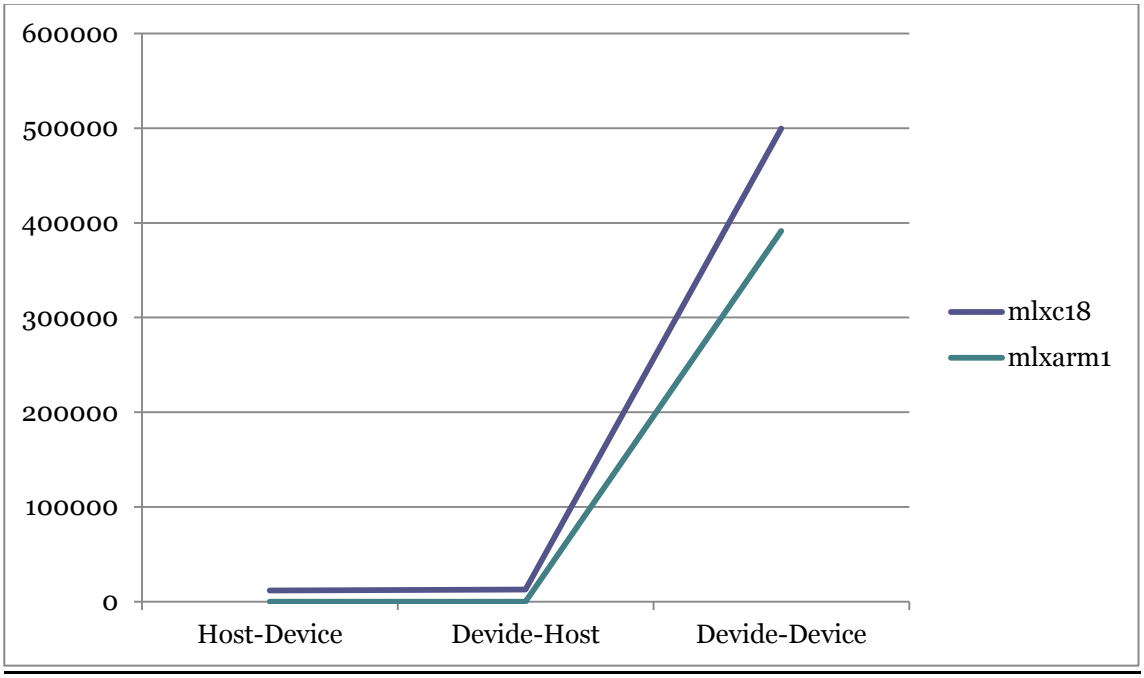

Figura 3.2.10 Comparativa test matrixMul en nodos proyecto

2.4 High Performance networks: Infiniband, RoCE

En este apartado vamos a hablar de las distintas tecnologías de red más comunes y actuales, mostrando sus prestaciones y sus ventajas actuales.

Infiniband

- Es un bus de comunicaciones serie de alta velocidad, de baja latencia y de baja sobrecarga de CPU, diseñado tanto para conexiones internas como externas.

- Es Bidireccional de tal manera que evita los problemas típicos asociados a buses paralelos en largas distancias.

- A pesar de ser una conexión serie, es muy rápido, ofreciendo una velocidad bruta de unos 2,5 Gigabits por segundo (Gbps) en cada dirección por enlace.

- También soporta doble e incluso cuádruples tasas de transferencia de datos, llegando a ofrecer 5 Gbps y 10 Gbps respectivamente

- Se usa una codificación 8B/10B, con lo que, de cada 10 bits enviados solamente 8 son de datos, de tal manera que la tasa de transmisión útil es 4/5 de la media.

- Teniendo esto en cuenta, los anchos de banda ofrecidos por los modos simple, doble y cuádruple son de 2, 4 y 8 Gbps respectivamente.

Caudal de Infiniband, bruto / eficaz			
	SDR	**DDR**	**QDR**
1X	2,5 / 2 Gbps	5 / 4 Gbps	10 / 8 Gbps
4X	10 / 8 Gbps	20 / 16 Gbps	40 / 32 Gbps
12X	30 / 24 Gbps	60 / 48 Gbps	120 / 96 Gbps

Figura 2.4.1 Prestaciones Infiniband

- Llega a ofrecer velocidades de hasta 2.0Gbps netos en cada dirección del enlace en un nodo simple, 4Gbps netos en un nodo doble y hasta 8Gbps netos en un nodo cuádruple.

- Estas enormes velocidades de conexión hacen que Infiniband sea una conexión con una muy importante presencia en superordenadores y clústers, por ejemplo, del top 500 de superordenadores, 226 están conectados internamente con Infiniband**,** 188 lo están con Gigabit Ethernet y el resto con Myrinet, Cray, Fat Tree u otras interconexiones a medida.

- Una de las principales ventajas de Infiniband sobre Ethernet es su bajísima latencia.

- Basándonos en los datos del estudio de Qlogic "Introduction to Ethernet Latency, an explanation to Latency and Latency measurement", la latencia en 10Gpbs Ethernet se sitúa en 5 microsegundos mientras que la de Infiniband se sitúa por debajo de los 3 microsegundos.

- Ethernet tiene una topología jerárquica, mientras Infiniband tiene una topología en la que cada nodo tiene una comunicación directa con cualquier otro nodo, permitiendo a la tarjeta de red leer o escribir datos directamente en otros servidores, evitándole este trabajo a los procesadores.

- En soluciones de virtualización, Infiniband también es mucho más eficiente que cualquier otra solución de conectividad**.**

RoCE

- RDMA sobre Converged Ethernet (RoCE).

- Es un protocolo de red que permite el acceso a la memoria directa remota (RDMA) a través de una red Ethernet.

- Hay dos versiones RoCE, RoCE v1 and RoCE v2.

- RoCE v1 es un protocolo de capa de enlace Ethernet y, por lo tanto, permite la comunicación entre dos hosts en el mismo dominio de difusión Ethernet.

- RoCE v2 es un protocolo de capa de internet en el cual los paquetes de RoCE v2, a diferencia de los de la versión 1, se pueden enrutar.

- Aunque el protocolo RoCE se beneficia de las características de una red Ethernet convergente, el protocolo también se puede usar en una red Ethernet tradicional o no convergente.

3. Aplicaciones

3.1 Descripción de aplicaciones usadas

En este punto se han instalado ciertas aplicaciones para comprobar las distintas prestaciones de los distintos procesadores.

En primer lugar se ha instalado la aplicación **Phoronix Test Suite** para poder comprobar las prestaciones del hardware de cada sistema (disco, procesador, memoria, sistema), de esta manera poder ver las diferencias entre las prestaciones de los distintos tipos de nodos.

Es una herramienta de benchmarking que suministra una gran cantidad de pruebas que abarcan las siguientes áreas: procesador, gráfica, red, sistema y rendimiento del disco. Es un software libre al estar publicado bajo la licencia GPLv3 y es multiplataforma, estando disponible para GNU/Linux, Windows, Mac (a día de hoy lo siguen llamando OS X en lugar de macOS), GNU/Hurd, Solaris y los sistemas operativos BSD.

Posteriormente hemos instalado **Relion – Master**, una aplicación real de cálculo de imagen 2D y 3D, con la cual podremos observar las diferencias entre usar uno o varios procesadores, así como las diferencias entre usar un nodo u otro o cómo usar la gpu local o la gpu remota gracias a rCUDA.

RELION (for REgularised LIkelihood OptimisatioN) es un programa informático real que emplea un enfoque Bayesiano empírico para refinar

(múltiples) reconstrucciones 3D o promedios de clase 2D en Cryo-microscopia electrónica (Cryo-EM)

Es desarrollado por el grupo de Sjors SCHERES en el laboratorio de MRC de biología molecular

3.2 Instalación

En este punto explicamos cómo hemos instalado estas aplicaciones y los correspondientes pasos que hemos seguido para llevar a cabo la instalación en los nodos.

Phoronix Test Suite

Proceso de instalación de la suite

El primer paso es descargar y descomprimir el fichero del repositorio oficial en nuestro nodo, para a continuación proceder con la instalación del benchmark para caracterizar los distintos nodos que me ha suministrado la universidad.

Como no tengo permiso de escritura en la ruta /usr para instalar software, tengo que instalar el phoronix en otra ruta en la cual, si tenga permisos para que mi usuario pueda instalar software, de esta manera he decidido instalarlo en mi home de alumno.

Para instalarlo nos dirigimos al directorio donde se encuentra el fichero que facilita la instalación y ejecutamos el siguiente comando:

./install-sh /home/alumno/phoronix

En este punto nos da un error en la instalación porque tenemos ciertas dependencias que tenemos que solventar e instalar antes de poder seguir con la instalación de la aplicación phoronix.

-Dependencia de PHP

Procedemos a instalar la dependencia de php en el nodo correspondiente para poder llevar a cabo la instalación de Phoronix, lo hemos instalado con el siguiente comando:

yum install php-cli php-xml

Instalación de los test de Phoronix

Vamos a lanzar los distintos tipos de test para caracterizar el hardware de cada nodo (disco, memoria, sistema), por lo que vamos instalar algunos de los test recomendados.

Los comandos que se han utilizado para instalarlos son los siguientes:

./phoronix-test-suite install pts/ramspeed

./phoronix-test-suite install pts/c-ray

./phoronix-test-suite install pts/schbench

./phoronix-test-suite install pts/tiobench

./phoronix-test-suite install pts/stream

./phoronix-test-suite install pts/cachebench

Relion-Master

Proceso de instalación Relion

En este apartado se va mostrar el proceso que se ha seguido para poder instalar la aplicación en nuestro nodo. Este proceso es más complejo que el de Phoronix.

Repositorio oficial de relion:
https://www2.mrc-lmb.cam.ac.uk/relion/index.php?title=Main_Page

-Descargar paquete instalación desde su github oficial:

wget https://codeload.github.com/3dem/relion/zip/master
mv master master.zip
unzip master

cd relion-master
mkdir build
cd build

-Exportar librerías correspondientes a nuestro path debido a las dependencias requeridas de MPI:

export PATH=/lib/intel/LIBS/MVAPICH2/2.3b/bin:$PATH

export LD_LIBRARY_PATH=/lib/intel/LIBS/MVAPICH2/2.3b/lib:$LD_LIBRARY_PATH

Compilación del programa:

cmake -DCMAKE_INSTALL_PREFIX=/bin/intel/RELION/relion-master-20180628/ -DGUI=OFF

make -j4

make install

cp -r /bin/arm/RELION/relion-master-20180628/external /bin/arm/RELION/relion-master-20180628/external

cd /bin/arm/RELION/relion-master-20180628/external/fftw/

*rm -rf fftw3**

-Instalación benchmarks:

https://www2.mrc-lmb.cam.ac.uk/relion/index.php?title=Benchmarks_%26_computer_hardware

-Descargar paquete de benchmarks desde su github oficial:

-wget ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion_benchmark.tar.gz .
-tar -xzf relion_benchmark.tar.gz
-cd relion_benchmark
-mkdir class2d
-mkdir class3d

3.3 Uso

En este apartado se va a observar cómo hemos ejecutado las aplicaciones anteriormente instaladas para pasar los test elegidos, además de comprobar las prestaciones de los nodos y ver sus distintos rendimientos y de esta manera poder observar las diferencias dependiendo de sus características.

Ejecución de los test de Phoronix

Para ejecutar un test de Phoronix, tenemos que acceder al directorio donde tenemos instalada la aplicación y lanzar el siguiente comando para cada uno de los test que queremos ejecutar en nuestro sistema:

cd /home/alumno/phoronuevo/bin/

./phoronix-test-suite benchmark pts/ramspeed

./phoronix-test-suite benchmark pts/c-ray

./phoronix-test-suite benchmark pts/schbench

./phoronix-test-suite benchmark pts/tiobench

./phoronix-test-suite benchmark pts/stream

./phoronix-test-suite benchmark pts/cachebench

Si hay algún test que da error por alguna razón la forma de poder reparar o depurar la instalación es con el comando:

./phoronix-test-suite debug-install pts/schbench

./phoronix-test-suite debug-bencharmk pts/schbench

Para ver la lista de test y suites disponibles podemos ejecutar este comando:

phoronix-test-suite list-available-tests

La lista de test recomendados por tipo podemos encontrarlo con la siguiente instrucción:

Phoronix-test-suite list-recommended-test

Ejecución de los test de Relion-Master

En este punto os mostramos como hemos lanzado las ejecuciones de los test del relion-master, hay test de ficheros 2d y de ficheros 3d, así tanto las ejecuciones que se usan la gpu, como que no se usa la gpu del nodo y solamente se usa la cpu. Se han cambiado algunos valores para que los test duren menor tiempo.

Para pasar los test hemos elegido un nodo con procesador arm, llamado nodo-arm y otro nodo con procesador Xeon, llamado nodo-x86.

Resumen de los test que se han realizado

- nodo-x86: Test CPU, GPU y como servidor de rCUDA

- nodo-arm: CPU y como cliente de rCUDA para las pruebas con GPU

Página oficial donde muestra las ejecuciones recomendadas para los distintos tipos de benchmark:

https://www2.mrc-lmb.cam.ac.uk/relion/index.php?title=Benchmarks_%26_computer_hardware

Test CPU y GPU del nodo nodo-x86

Para poder ejecutar los test se tienen que ejecutar las siguientes instrucciones:

*export PATH=/*home/alumno/*/tfg/relion-master-carlos/x86_64/bin: /lib/intel/LIBS/MVAPICH2/2.3b/bin:/lib/intel/CUDA/8.0/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/ibutils/bin*

export LD_LIBRARY_PATH=/home/alumno/*/tfg/relion-master-carlos/x86_64/lib:*/home/alumno/*tfg/relion-master-carlos/x86_64/external/fftw/lib:/lib/intel /LIBS/MVAPICH2/2.3b/lib:/lib/intel/CUDA/8.0/lib64:/lib/intel /LIBS/CUDNN/8.0-v5.1/lib64*

cd /home/alumno/*/tfg/relion-master-carlos/x86_64/benchmarks/relion_benchmark*

Comandos para ejecutar los test CLASS2D y CLASS3D

CLASS2D-CPU-nodo-x86

time mpirun -n 24 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ctf --iter 1 --tau2_fudge 2 --particle_diameter 30 --K 2 --zero_mask --oversampling 1 --psi_step 6 --offset_range 5 --offset_step 2 --norm --scale --dont_combine_weights_via_disc --random_seed 0 --o class2d

CLASS2D-GPU-nodo-x86

time mpirun -n 2/4/8/16/24 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ctf --iter 1 --tau2_fudge 1 --particle_diameter 30 --K 2 --zero_mask --oversampling 1 --psi_step 1 --offset_range 1 --offset_step 1 --norm --scale --dont_combine_weights_via_disc --random_seed 0 --o class2d –gpu

CLASS3D-CPU-nodo-x86

time mpirun -n 2/4/8/16/24 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 25 --ctf --ctf_corrected_ref --iter 1 --tau2_fudge 1 --particle_diameter 30 --K 2 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 1 --offset_range 1 --offset_step 1 --sym C1 --norm –scale --dont_combine_weights_via_disc --random_seed 0 --o class3d

CLASS3D-GPU-nodo-x86

time mpirun -n 2/4/8/16/24 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 25 --ctf --ctf_corrected_ref --iter 1 --tau2_fudge 1 --particle_diameter 30 --K 2 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 1 --offset_range 1 --offset_step 1 --sym C1 --norm –scale --dont_combine_weights_via_disc --random_seed 0 --o class3d --gpu

Test CPU del nodo nodo-arm

Para poder ejecutar los test se tienen que ejecutar las siguientes instrucciones:

export PATH=/home/alumno/*tfg/relion-master-carlos/aarch64/bin:*
/lib/arm/LIBS/MVAPICH2/2.3b/bin:/lib/arm/CUDA/8.0/bin:/usr/local/bin:/usr/bi
n:/usr/local/sbin:/usr/sbin:/opt/ibutils/bin

export LD_LIBRARY_PATH=/home/alumno/*tfg/relion-master-*
carlos/aarch64/lib: /home/alumno/*tfg/relion-master-*
carlos/aarch64/external/fftw/lib:/lib/arm/LIBS/MVAPICH2/2.3b/lib:/lib/arm
/CUDA/8.0/lib64:/lib/arm/LIBS/CUDNN/8.0-v5.1/lib64

cd /home/alumno/*tfg/relion-master-*
carlos/aarch64/benchmarks/relion_benchmark

Comandos para ejecutar los test CPU de CLASS2D y CLASS3D

CLASS2D-CPU-nodo-arm

time mpirun -n 48 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ctf --iter 1 --tau2_fudge 2 --particle_diameter 30 --K 2 --zero_mask --oversampling 1 --psi_step 6 --offset_range 5 --offset_step 2 --norm --scale --dont_combine_weights_via_disc random_seed 0 --o class2d

CLASS3D-CPU-nodo-arm

time mpirun -n 2/4/8/16/24/32/48 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 25 --ctf --ctf_corrected_ref --iter 1 --tau2_fudge 1 --particle_diameter 30 --K 2 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 1 --offset_range 1 --offset_step 1 --sym C1 --norm –scale --dont_combine_weights_via_disc --random_seed 0 --o class3d

Test GPU del nodo nodo-arm

En este punto es donde se ha usado la tecnología rCUDA, ya que el nodo nodo-x86 actúa como servidor y el nodo nodo-arm como cliente, es decir el nodo nodo-arm utilizara la GPU del nodo nodo-x86 para pasar los test 3D de Relion

Instrucciones ejecutadas para la ejecución del servidor nodo-x86 rCUDA**:**

export LD_LIBRARY_PATH=/lib/intel/CUDA/8.0/lib64:/lib/intel /LIBS/CUDNN/8.0-v6.0/lib64

cd /home/alumno/*tfg/rCUDAv18.03beta-CUDA8.0-cuDNN6.0/x86_64/bin*

./rCUDAd –iv

Instrucciones ejecutadas para la ejecución de relion en el cliente nodo-arm**:**

export PATH=/home/alumno/*tfg/relion-master-carlos/aarch64/bin: /lib/arm/LIBS/MVAPICH2/2.3b/bin:/lib/arm/CUDA/8.0/bin:/usr/local/bin:/usr/bi n:/usr/local/sbin:/usr/sbin:/opt/ibutils/bin*

export LD_LIBRARY_PATH=/home/alumno/*tfg/relion-master-carlos/aarch64/lib:*/home/alumno/*tfg/relion-master-carlos/aarch64/external/fftw/lib:/lib/arm/LIBS/MVAPICH2/2.3b/lib:*/home/alum no/*tfg/rCUDAv18.03beta-CUDA8.0-cuDNN6.0/aarch64/lib*

export RCUDA_DEVICE_COUNT=1

export RCUDA_DEVICE_0=nodo-x86

cd /home/alumno/*tfg/relion-master-carlos/aarch64/benchmarks/relion_benchmark*

Comandos para ejecutar los test GPU de CLASS2D y CLASS3D

CLASS2D-GPU-nodo-arm

time mpirun -n 2/4/8/16/24/32/48 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ctf --iter 1 --tau2_fudge 1 --particle_diameter 30 --K 2 --zero_mask --oversampling 1 --psi_step 1 --offset_range 1 --offset_step 1 --norm --scale --dont_combine_weights_via_disc --random_seed 0 --o class2d --gpu

CLASS3D-GPU-nodo-arm

time mpirun -n 2/4/8/16/24/32/48 `which relion_refine_mpi` --i Particles/shiny_2sets.star --ref emd_2660.map:mrc --firstiter_cc --ini_high 25 --ctf --ctf_corrected_ref --iter 1 --tau2_fudge 1 --particle_diameter 30 --K 2 --flatten_solvent --zero_mask --oversampling 1 --healpix_order 1 --offset_range 1 --offset_step 1 --sym C1 --norm –scale --dont_combine_weights_via_disc --random_seed 0 --o class3d –gpu

4. Resultados de prestaciones

4.1 Phoronix Test Suite

	nodo-arm	**mlxf1**	**nodo-x86**
c-ray (seg)	13,83	14,48	7,11
Cachebench (mb/s)	- Read: 3562,15 - Write: 5034,31	- Read: 2187,97 - Write: 17384,36	- Read 2625,72 - Write: 22012,47
Schbench (usec)		-1 Variante: 234 -2 Variante: 243	- 1 Variante: 87 - 2 Variante: 90
Tiobench (mb/s)	-1 Variante: 9,83 -2 Variante: : 104,71	-1 Variante: 12,35 -2 Variante: 87,37	- 1 Variante: 13.44 - 2 Variante: 16208,40
Stream (mb/s)		-Copy: 59821 -Add: 48806,80	- Copy: 75955,78 - Add: 73307,18
Ramspeed (mb/s)		-Copy-Integer :18,833	- Copy-Integer: 21045,23 - Copy-Float: 20988,51 - Add-Integer: 22538,97

Figura 4.1 Resultados Test Phoronix nodo-arm, mlxfm1, nodo-x86

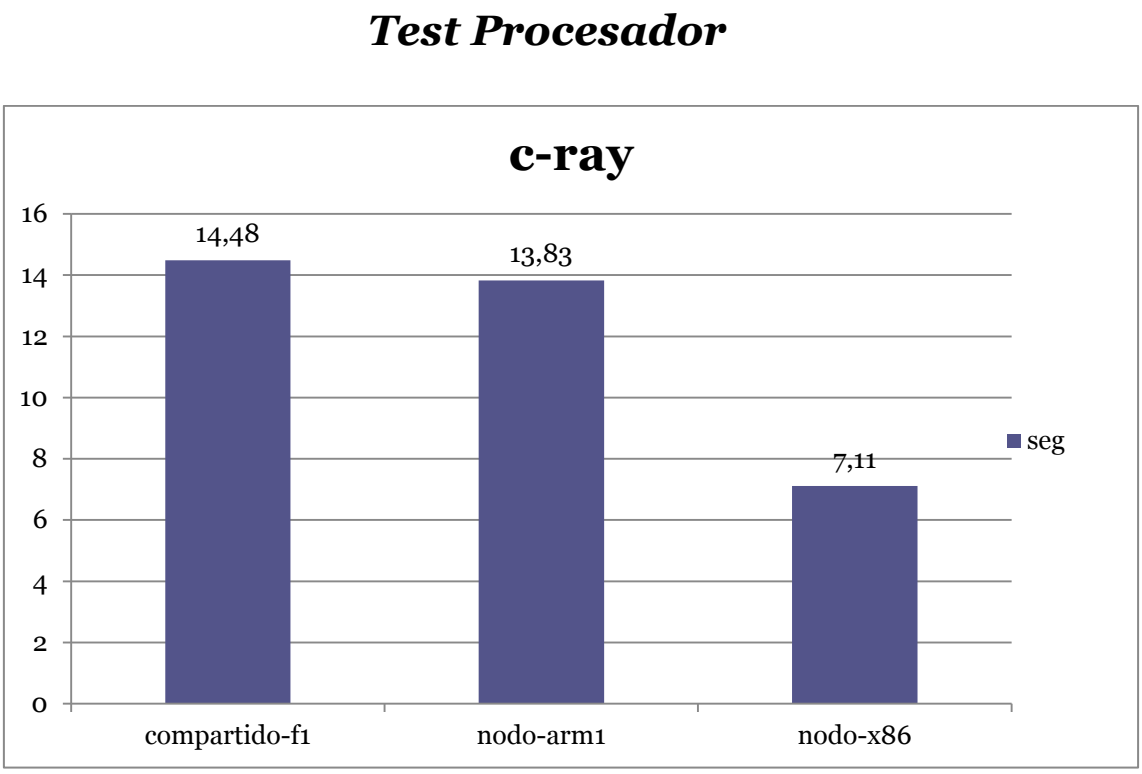

Figura 4.2 Resultados Test Phoronix c-ray 3 nodos

Se observa que el nodo nodo-x86 con el procesador Xeon tiene un tiempo de ejecución menor a los dos nodos restantes, los cuales uno es de bajo consumo (nodo-arm) y otro es el nodo compartido f1. Se observa que el nodo nodo-x86 es mucho más rápido que los demás.

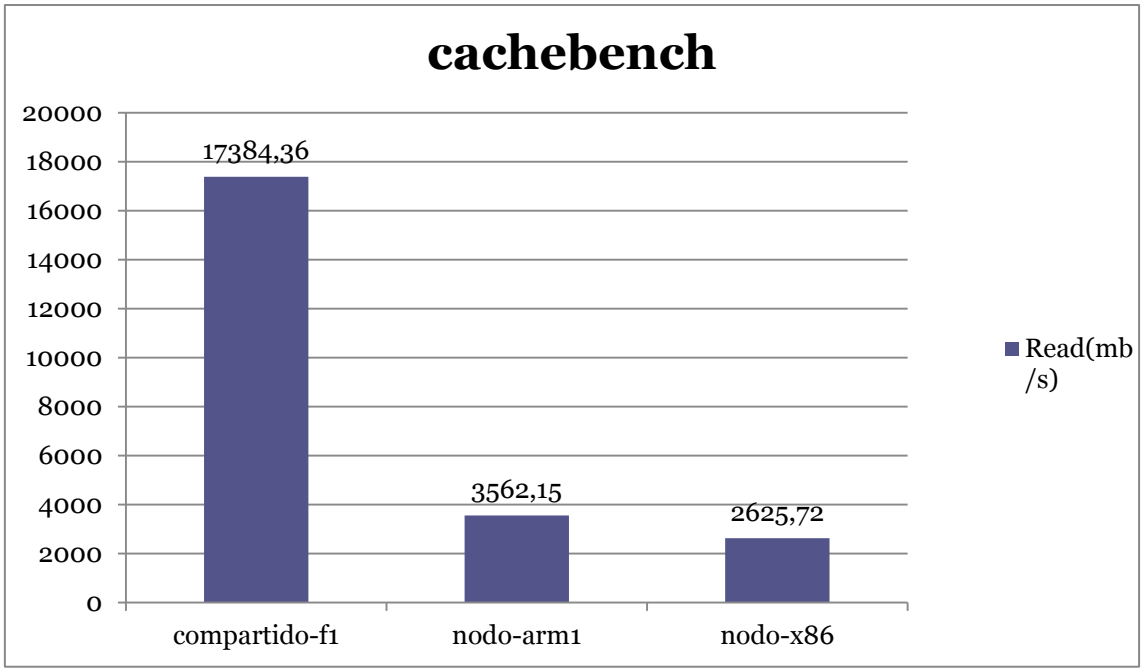

Figura 4.3 Resultados Test Phoronix cachebench-read 3 nodos

1 Variante. Read

Cuanto mayor es la transferencia mejor es el resultado, se observa que el nodo compartido-f1 tiene la mejor prestación en lectura.

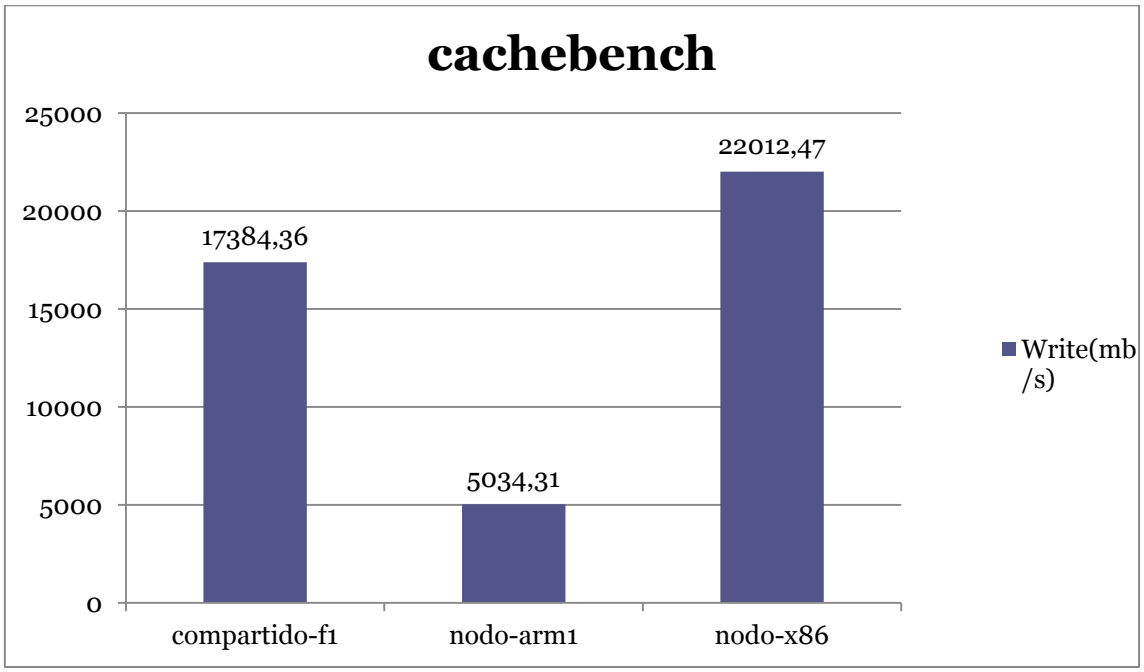

Figura 4.4 Resultados Test Phoronix cachebench-write 3 nodos

2 Variante. Write

Cuanto mayor es la transferencia mejor es el resultado, se observa que el nodo nodo-x86 tiene las mejores prestaciones en escritura.

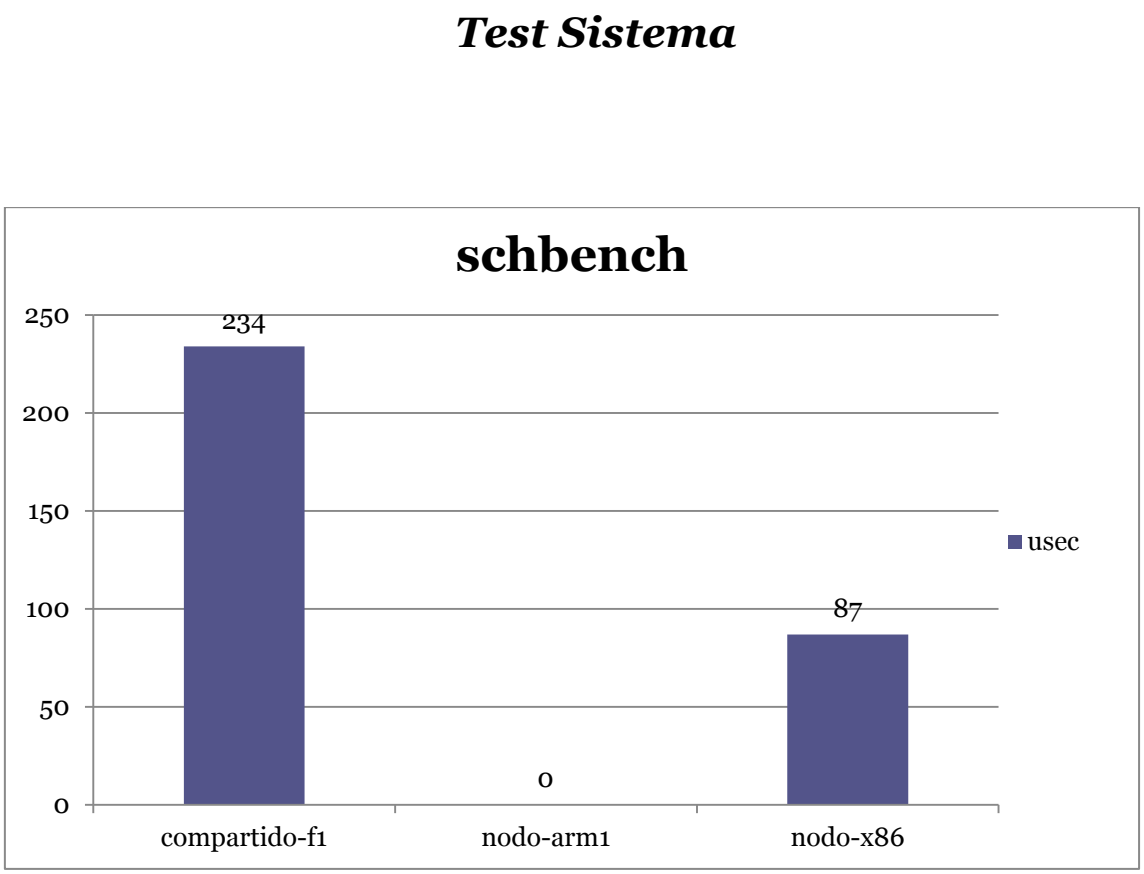

Figura 4.5 Resultados Test Phoronix schbench-4-4 3 nodos

***1 Variante:* 4 Message threads -4 workers/thread**

Observamos que cuanto menor es el valor de usec, mejor prestación ofrece el sistema, donde podemos observar que el nodo nodo-x86 tiene unas mejores prestaciones

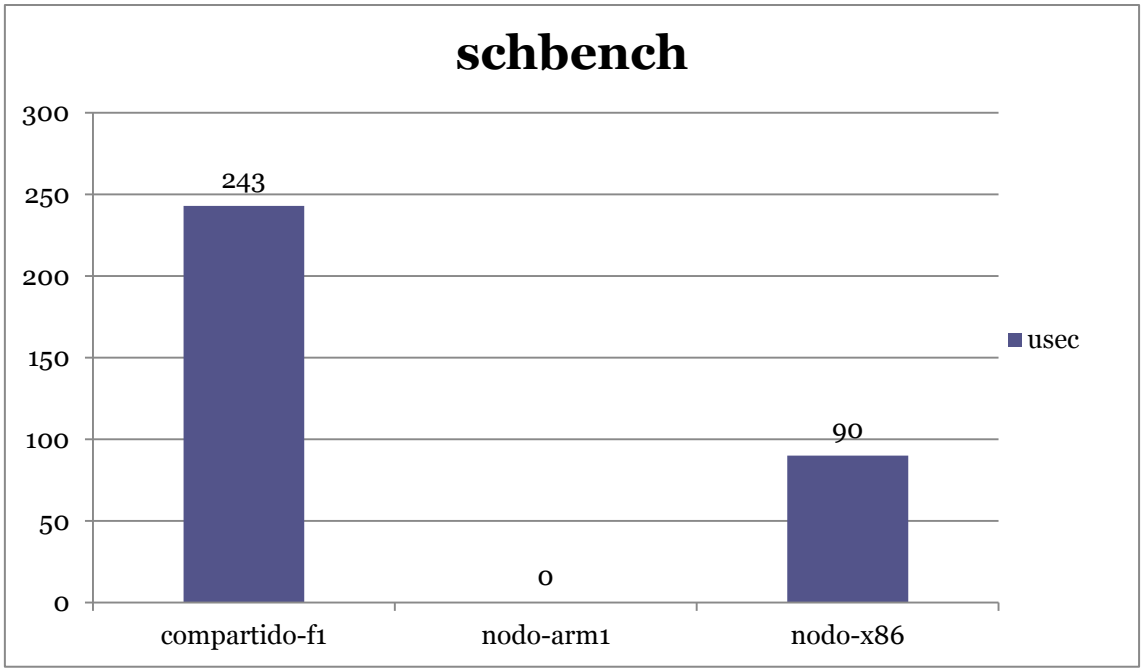

Figura 4.6 Resultados Test Phoronix schbench-2-2 3 nodos

2 Variante 2: Message threads -2 workers/thread

Observamos que cuanto menor es el valor de usec, mejores prestaciones ofrece el sistema, donde podemos observar que el nodo nodo-x86 también tiene unas mejores prestaciones con otros parámetros (menos mensajes y menos workers)

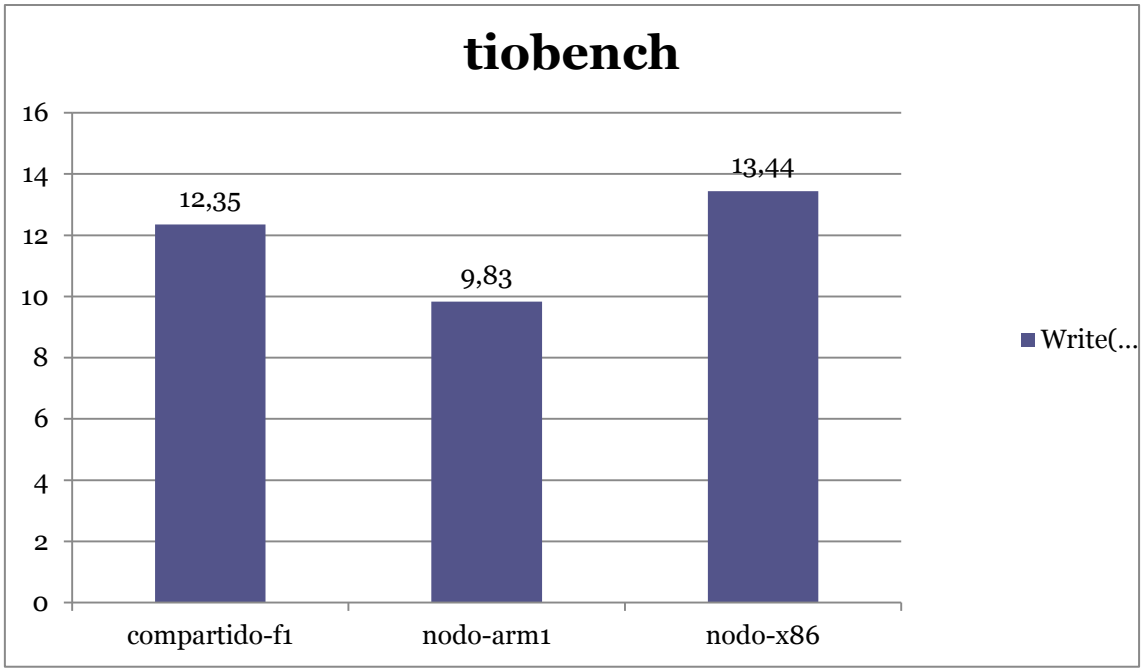

Figura 4.7 Resultados Test Phoronix tiobench-w-32-4 3 nodos

1 Variante: Write – Tamaño Hilo: 32mb – Numero de Hilos: 4

Observamos que cuanto más grande es el valor de la transferencia, mejores prestaciones ofrece el sistema, donde podemos observar que el nodo nodo-x86 tiene unas mejores prestaciones que los otros dos testeados.

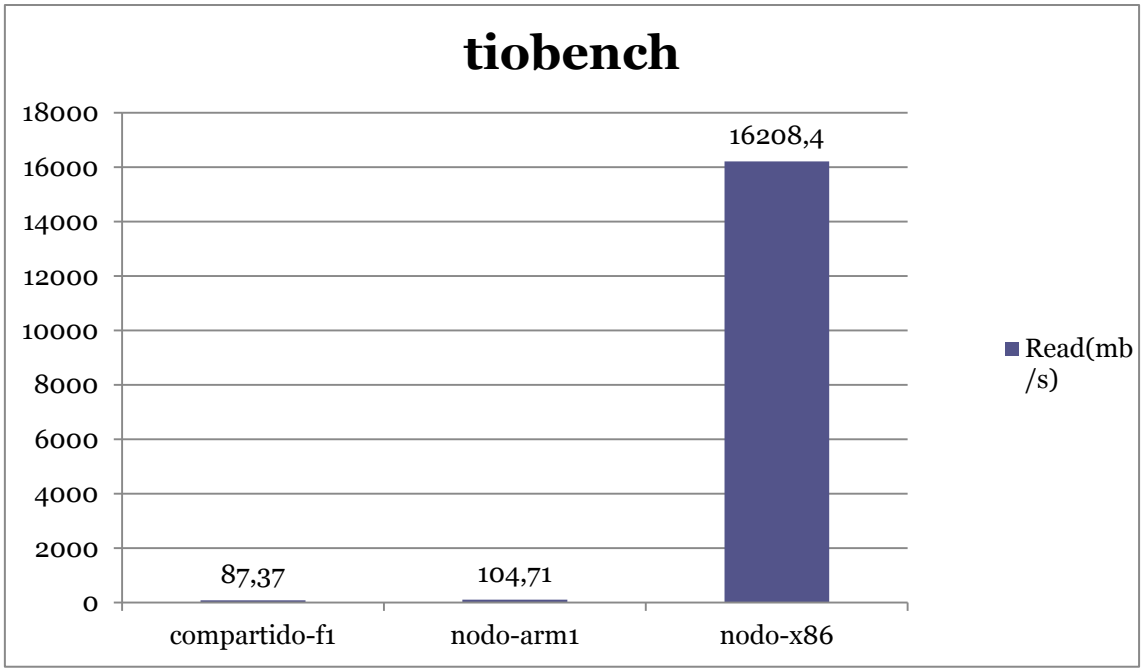

Figura 4.8 Resultados Test Phoronix tiobench-r-64-8 3 nodos

2 Variante: Read – Tamaño Hilo: 64mb – Numero de Hilos: 8 (Revisar mlxa8)

Observamos que cuanto más grande es el valor de la transferencia, mejores prestaciones ofrece el sistema, donde podemos observar que el nodo nodo-x86 tiene unas mejores prestaciones que los otros dos testeados.

Test Disco

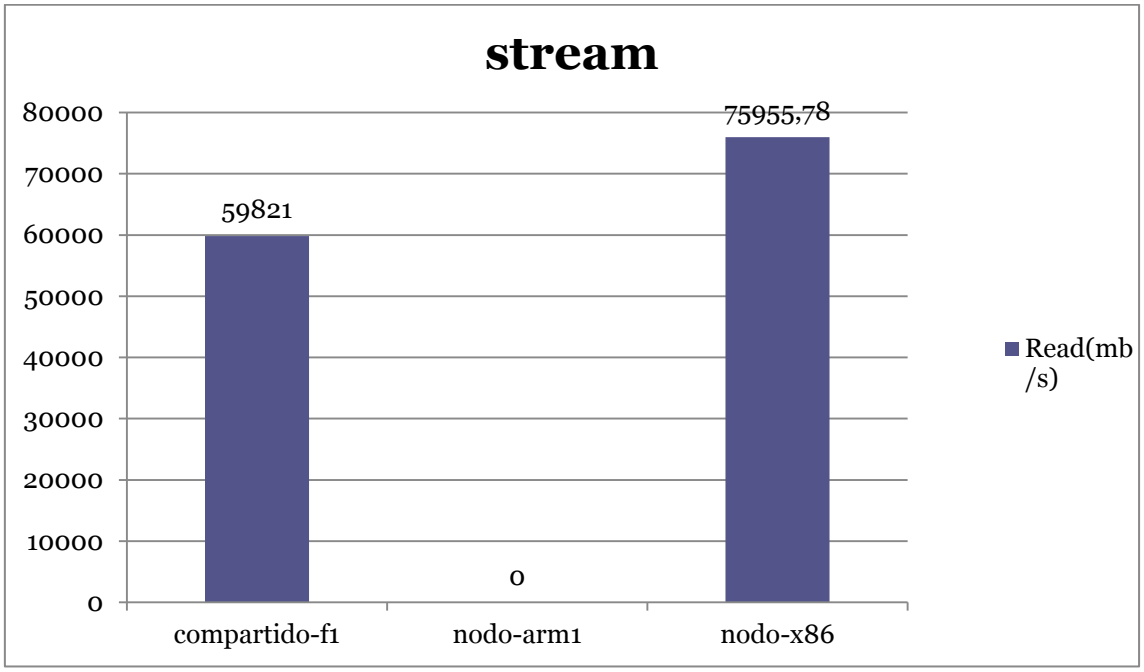

Figura 4.9 Resultados Test Phoronix stream-copy 3 nodos

1 Variante: Copy

Observamos que cuanto mayor es la transferencia mejores prestaciones tiene el nodo, por lo que vemos que el nodo nodo-x86 tiene unas prestaciones superiores.

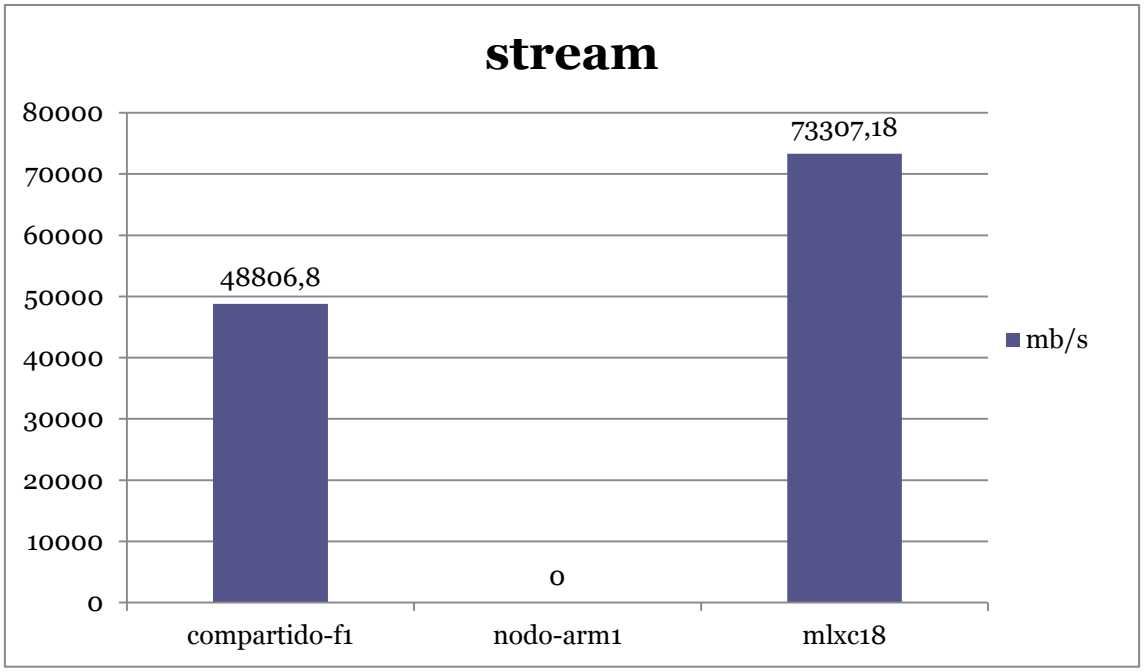

Figura 4.10 Resultados Test Phoronix stream-add 3 nodos

2 Variante: Add

Observamos que cuanto mayor es la transferencia mejores prestaciones tiene el nodo, por lo que vemos que el nodo nodo-x86 tiene unas prestaciones superiores.

Test Ram

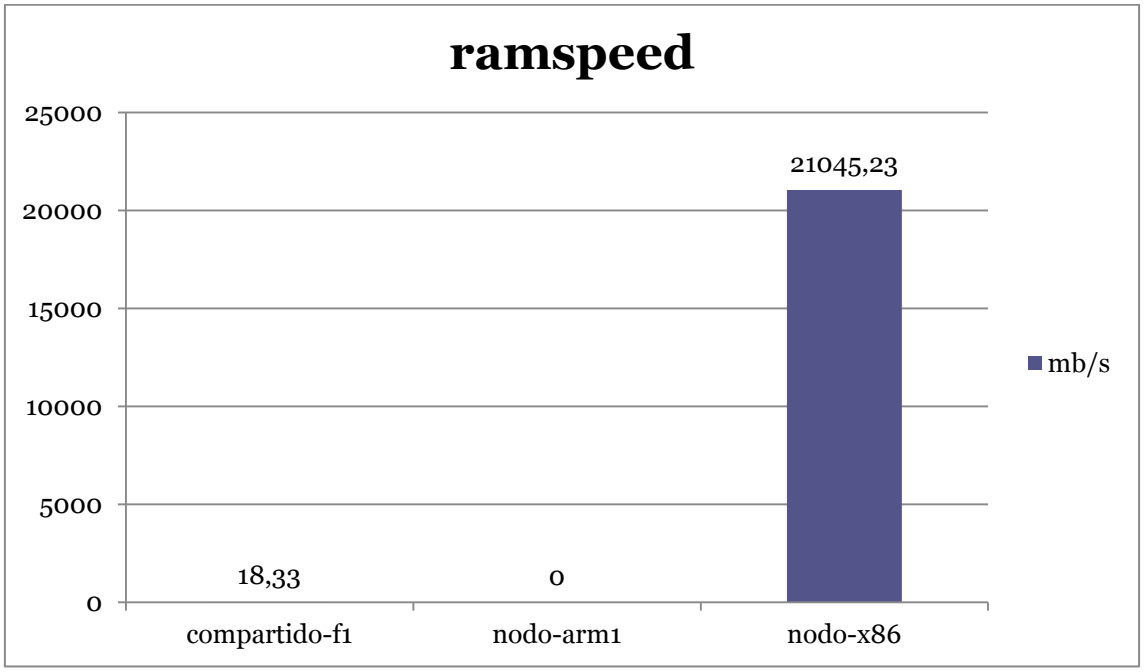

Figura 4.11 Resultados Test Phoronix ramspeed 3 nodos

1 Variante: Copy - Integer

Observamos también que cuanto mayor es la transferencia, mejores prestaciones tiene el nodo, por lo que vemos que el nodo nodo-x86 tiene unas prestaciones muy superiores, al ser un nodo no compartido.

4.2 Relion-Master

Tiempos de ejecución de los test en relion-master en nodo-x86:

nodo-x86	nodo-x86-2	nodo-x86-4	nodo-x86-8	nodo-x86-16	nodo-x86-20
CLASS2D	402	152	75	34	34
CLASS2D-GPU	43	19	17	16	19
CLASS3D	287	108	62	33	35
CLASS3D-GPU	62	24	24	15	20

Figura 4.2.1 Tabla resultados Relion-Master

Aquí os mostramos los resultados representados en una gráfica.

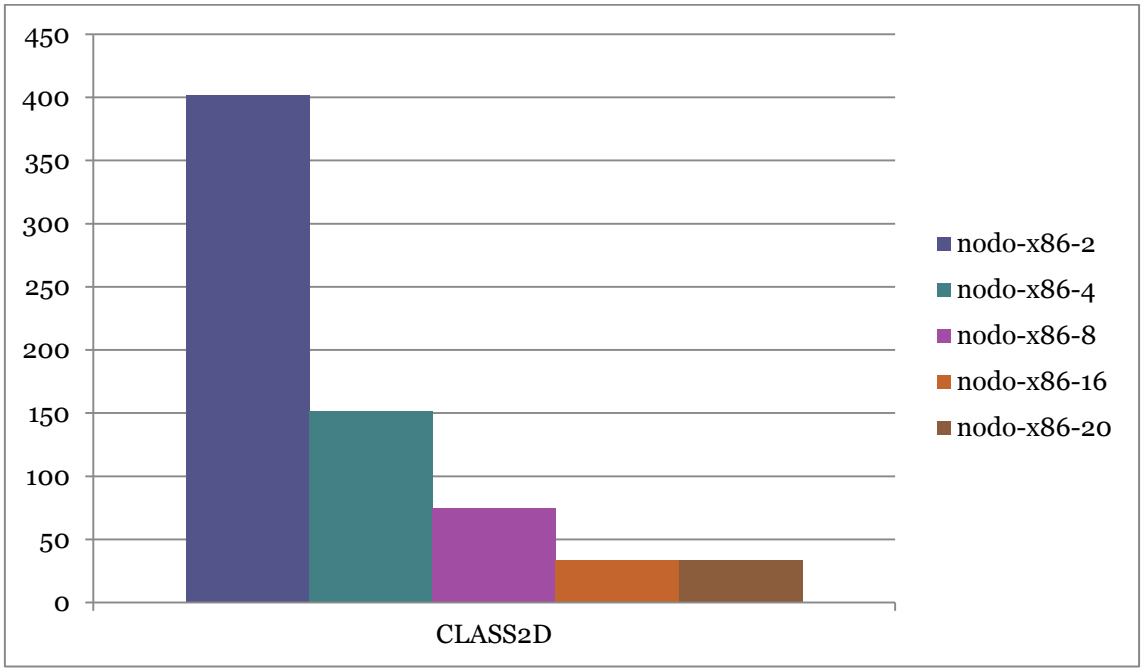

Figura 4.2.2 Resultados Relion-Master nodo-x86 Class2D

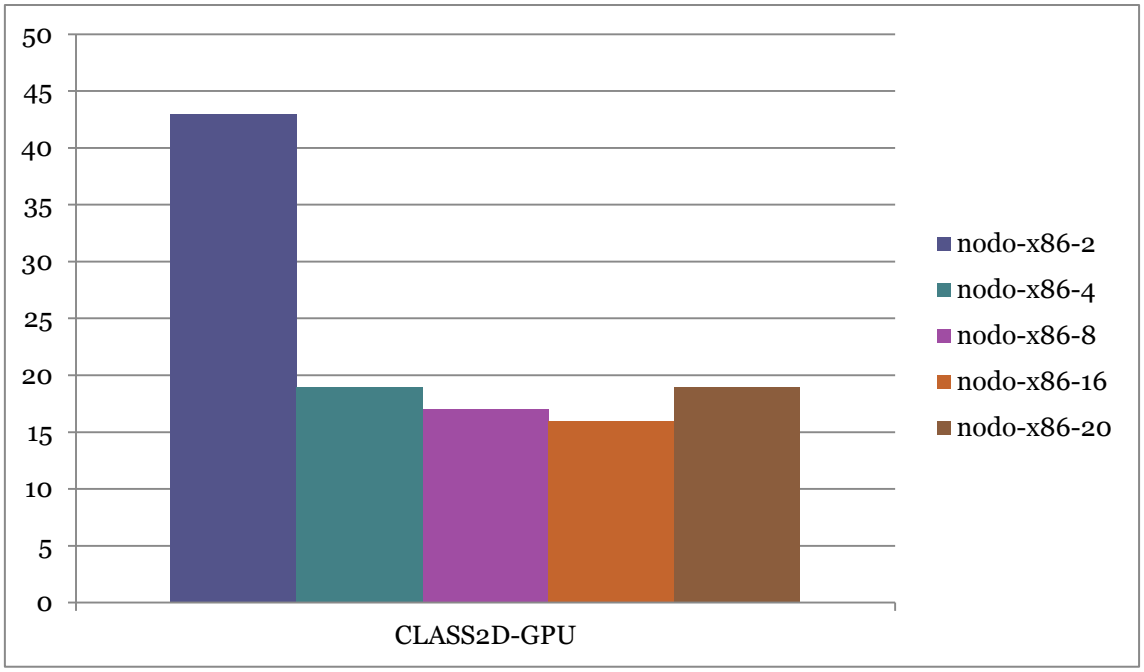

Figura 4.2.3 Resultados Relion-Master nodo-x86 Class2D-GPU

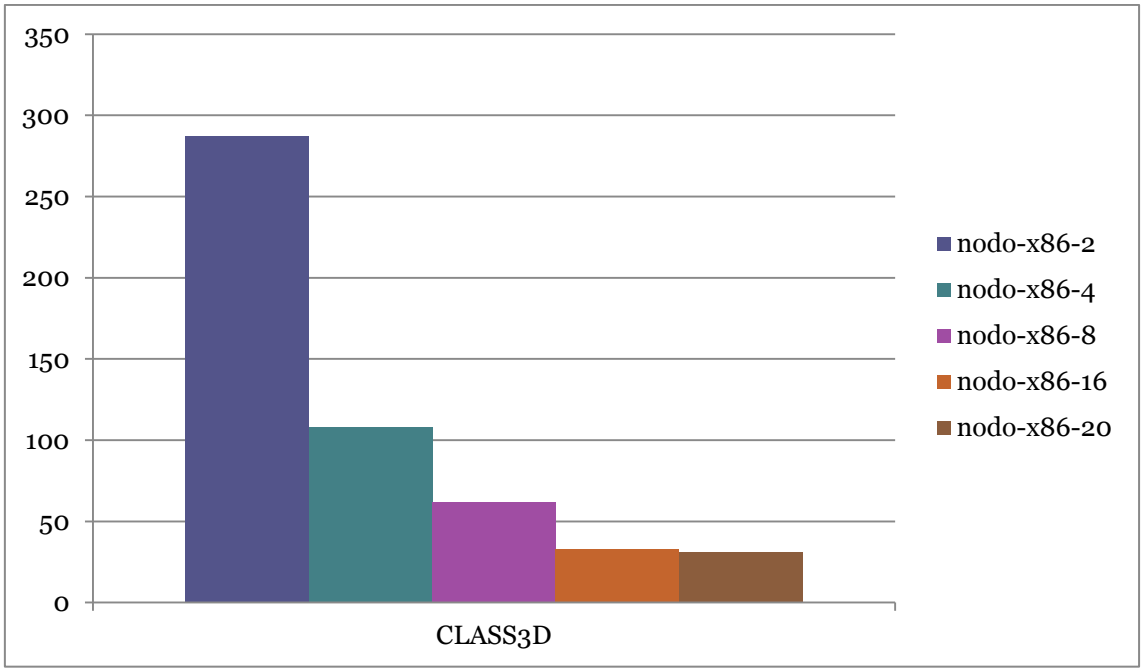

Figura 4.2.4 Resultados Relion-Master nodo-x86 Class3D

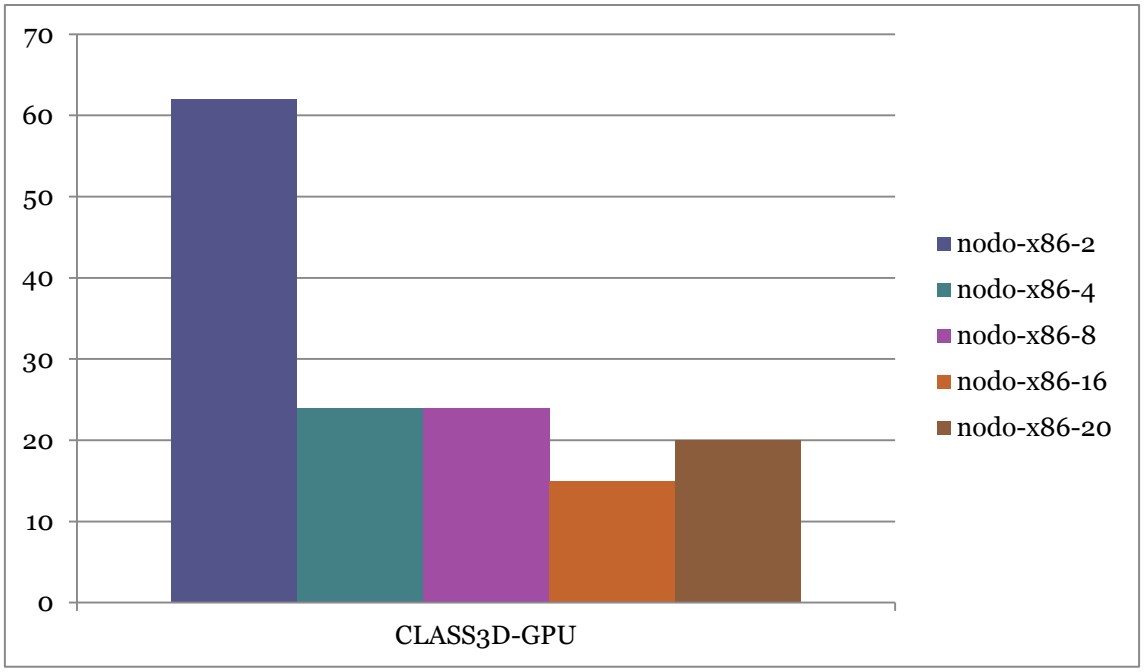

Figura 4.2.5 Resultados Relion-Master nodo-x86 Class3D-GPU

Resultados ejecución relion-master en nodo-arm:

nodo-arm	nodo-arm-2	nodo-arm-4	nodo-arm-8	nodo-arm-16	nodo-arm-24	nodo-arm-32	nodo-arm-48
CLASS2D	2517	848	362	182	127	109	74
CLASS2D-GPU	581	207	101	141	136	143	*
CLASS3D	1399	475	211	106	79	69	64
CLASS3D-GPU	1100	325	210	106	84	67	56

Figura 4.2.6 Tabla resultados Relion-Master

Aquí os mostramos los resultados representados en una gráfica.

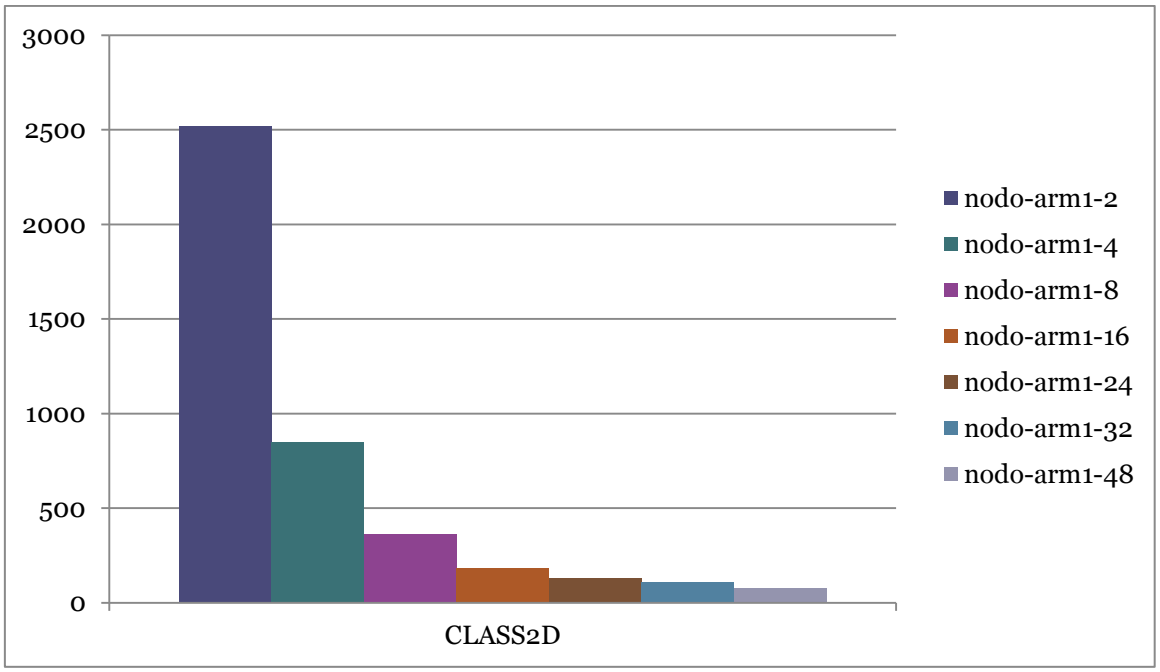

Figura 4.2.7 Resultados Relion-Master nodo-arm Class2D

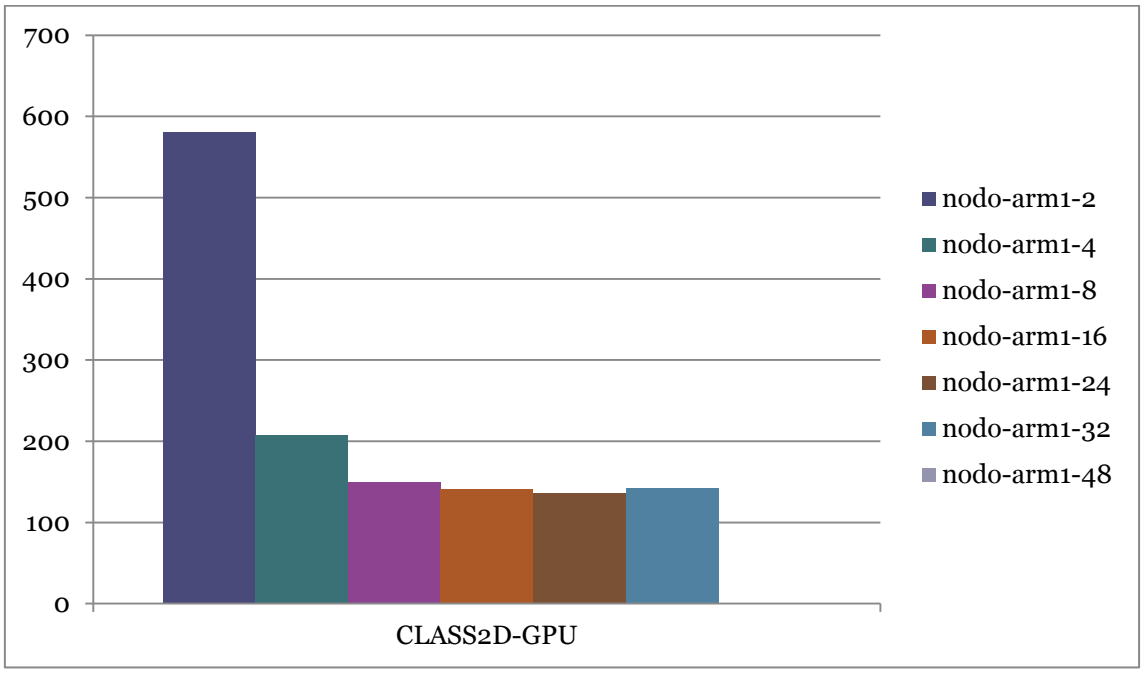

Figura 4.2.8 Resultados Relion-Master nodo-arm Class2D-GPU

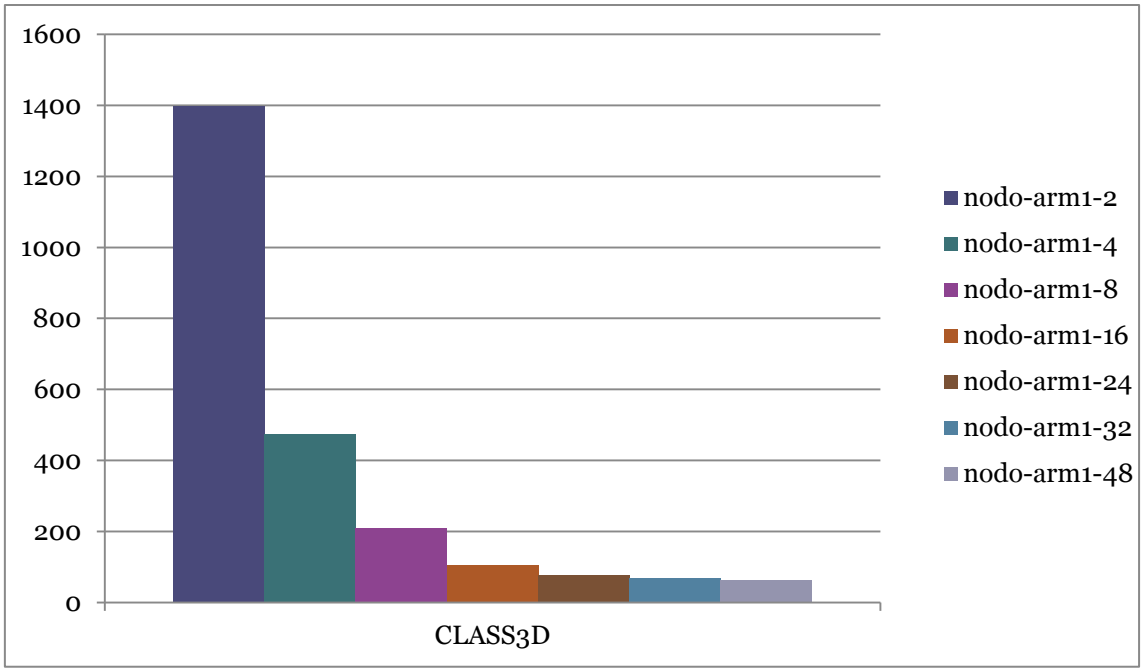

Figura 4.2.9 Resultados Relion-Master nodo-arm Class3D

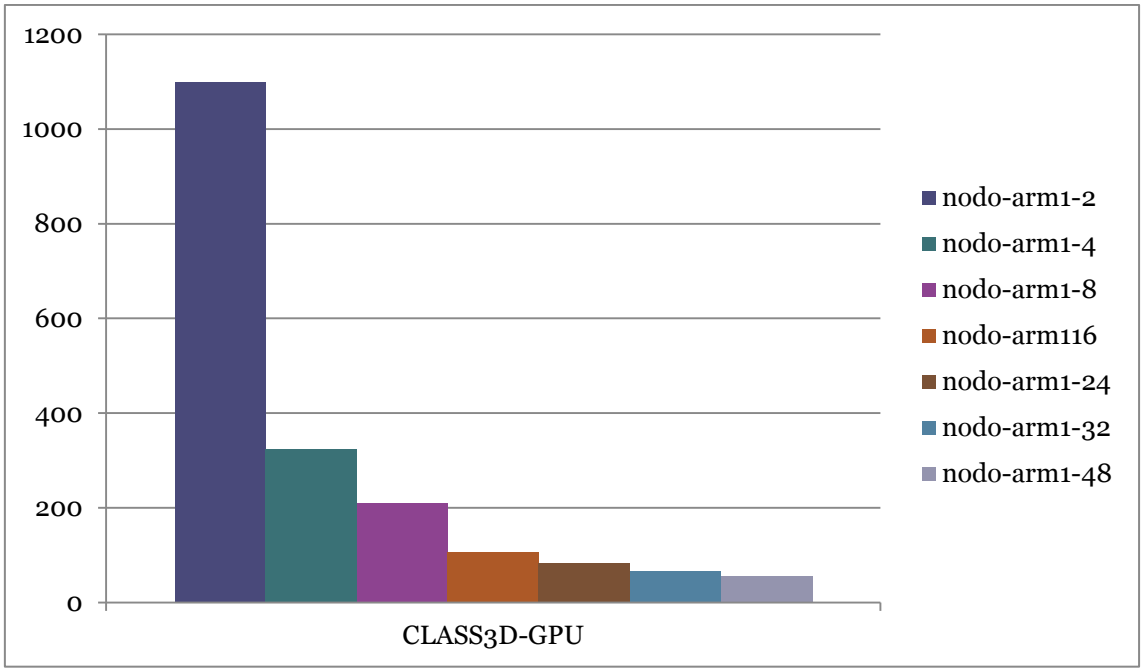

Figura 4.2.10 Resultados Relion-Master nodo-arm Class3D-GPU

*Observamos que el test de Class3d-gpu + rCUDA con 48 CPU, ha proporcionado un error de ejecución. Este error se debe a que la GPU no tiene memoria suficiente para dar servicio a 48 CPUs que requiere el nodo nodo-arm.

5. Conclusiones

5.1 Resumen

En este punto se va a hablar de las conclusiones que se han obtenido en este proyecto.

Phoronix se ha ejecutado para ver qué diferencias existen en cuanto a prestaciones entre los dos nodos a investigar (nodo-arm y nodo-x86), y ver las diferencias que existen entre un nodo de bajo consumo y uno que no lo es. Phoronix es un test artificial, por lo que más adelante durante el proyecto se ha usado Relion, ya que esta, es una aplicación real, y la hemos usado para ver si las conclusiones del test artificial se confirman cuando usamos una aplicación real.

Se puede observar que el ancho de banda con CUDA es muy superior ya que las transferencias se realizan mediante la interfaz pci-express, a su vez el ancho de banda con rCUDA es mucho menor porque las transferencias se hacen por la red (en este caso 1GbE).

Relion –Master

NODO-X86:

- Se observa, que lógicamente cuantos menos CPU son ejecutados (tanto en los test class2d y class3d) el tiempo de ejecución es superior, y a su vez conforme vamos aumentando el número de CPUs a utilizar, el test tiene un tiempo de ejecución inferior.

- Test class2d-gpu y class3d-gpu: En la ejecución de estos dos test durante la versión GPU, podemos observar que tarda mucho menos que la versión en CPU, debido al aumento de rendimiento por usar la tarjeta gráfica P100, que tiene integrada este nodo.

- Por otro lado, cuando en los test class2d-GPU, compartimos la única GPU que tiene el nodo nodo-x86 entre 4 CPUs, el tiempo es la mitad que cuando la compartimos con 2 CPUs. A partir de ahí, compartir la única GPU entre más de 4 CPUs no produce una mejora muy notable. Seguramente es debido a que el uso de la GPU esté llegando al límite de sus capacidades. Prueba de ello es que con 20 CPUs el tiempo incluso se incrementa un poco.

- Para finalizar durante los test class3d-gpu vemos que pasa algo parecido que en los test class2d-gpu, cuando se comparte la única GPU que tiene este nodo entre 4 CPU, el tiempo también es más de la mitad que cuando lo compartimos entre 2 CPU. A partir de ahí compartir la GPU entre más CPU no produce ninguna mejora notable en líneas generales, incluso con 20 CPU el tiempo se incrementa un poco, probablemente al llegar al límite de las capacidades de la GPU.

NODO-ARM:

- Debido a que el nodo nodo-arm es un nodo de bajo consumo, podemos observar en los test Class2d y Class3d que las prestaciones son inferiores y los tiempos de ejecución de los test son superiores respecto al nodo nodo-x86 (nodo que a su vez tiene un consumo superior debido a su procesador Xeon, debido a esto tiene un mayor consumo lo cual hace que tenga unas mejores prestaciones).

-Igual que en el nodo nodo-x86 a medida que aumentamos el número de procesadores a usar durante los test class2d y class3d el tiempo de ejecución es menor.

-Durante los test class2d-gpu y class3d-gpu vemos que cuando se utilizan pocas cpu la diferencia respecto a los test class2d/3d es mayor, pero conforme aumentamos el número de cpu y la GPU va a llegando al límite de sus capacidades, el tiempo se va igualando e incluso observamos que es superior utilizando la GPU, esto es debido a las transferencias y comunicaciones por la interfaz de red, ya que estas son más lentas y este factor penaliza el tiempo general de ejecución.

Por lo que, a pesar de los beneficios informados por la virtualización remota de la GPU, su uso también presenta algunos inconvenientes. La principal preocupación de esta técnica es que se accede a las GPU a través de una red, lo que tradicionalmente ha reducido el rendimiento de los movimientos de datos desde / hacia la GPU con respecto al caso local. La razón principal del bajo rendimiento es que con las GPU locales, los datos tienen que pasar solo a través del enlace PCIe local. Por el contrario, con control remoto, los datos de las GPU deben atravesar el enlace PCIe local, toda la red, y el enlace PCIe remoto. Además, dado que el ancho de banda de la red ha sido tradicionalmente menor que el del enlace PCIe, el rendimiento al usar control remoto GPUs ha sido tradicionalmente penalizado

5.2 Opinión Personal

El desarrollo de este proyecto me ha servido para conocer y profundizar sobre la virtualización de GPU dentro de una red local, además de conocer algunas de los distintos tipos de arquitecturas y procesadores actuales.

En cuanto al resultado del proyecto estoy contento, ya cumple con el alcance inicial establecido. Aunque en ciertos momentos he encontrado más dificultades para avanzar con el desarrollo, debido al tiempo de ejecución de los distintos tipos de test lanzados en los nodos. Pero al final, he podido finalizar dentro de los plazos establecidos.

Sería interesante hacer este TFG de investigación con la infraestructura y tecnología INFINIBAND, donde los resultados serían mejores, debido a sus mejores prestaciones, ente ellas, el ancho de banda.

Por último, dar las gracias a mis tutores por guiarme con las dudas que me iban surgiendo y por ayudarme a perfilar los detalles del proyecto.

6. Bibliografía

1. https://es.wikipedia.org/wiki/Intel_Xeon
2. https://www.researchgate.net/profile/Adrian_Castello/publication/266209971_Acelerando_Aplicaciones_Cientificas_con_GPUs_Remotas_y_Procesadores_de_Bajo_Consumo/links/542aa2da0cf29bbc1267bb43/Acelerando-Aplicaciones-Cientificas-con-GPUs-Remotas-y-Procesadores-de-Bajo-Consumo.pdf?origin=publication_detail
3. https://www.researchgate.net/profile/Adrian_Castello/publication/266209971_Acelerando_Aplicaciones_Cientificas_con_GPUs_Remotas_y_Procesadores_de_Bajo_Consumo/links/542aa2da0cf29bbc1267bb43/Acelerando-Aplicaciones-Cientificas-con-GPUs-Remotas-y-Procesadores-de-Bajo-Consumo.pdf?origin=publication_detail
4. http://www.nvidia.es/object/tesla-supercomputer-workstations-es.html
5. https://www.nvidia.es/object/tesla-k20-gpu-supercomputing-20121112-es.html
6. https://www.nvidia.es/object/cuda-parallel-computing-es.html
7. https://www.nvidia.es/object/tesla-p100-es.html
8. https://la.nvidia.com/object/tesla-p100-la.html
9. https://es.wikipedia.org/wiki/CUDA
10. https://phoronix-test-suite.com/documentation/phoronix-test-suite.html
11. https://phoronix-test-suite.com/documentation/phoronix-test-suite.html#%0AGettingStarted%0A
12. https://www.xataka.com/componentes/la-super-tesla-k20-de-nvidia-esta-aqui
13. http://infyseg.blogspot.com/2013/11/nvidia-tesla-k40-la-gpu-mas-rapida-del.html
14. https://www.muycomputer.com/2017/04/10/nvidia-lanza-la-tesla-p100/
15. https://en.wikipedia.org/wiki/RDMA_over_Converged_Ethernet
16. https://www2.mrc-lmb.cam.ac.uk/relion/index.php?title=Benchmarks_%26_computer_hardware#Getting_started

17. https://wiki.ubuntu.com/PhoronixTestSuite
18. https://es.wikipedia.org/wiki/InfiniBand
19. https://es.wikipedia.org/wiki/Intel_Atom
20. https://cyberseguridad.net/index.php/280-que-es-infiniband
21. https://hardzone.es/2018/03/10/procesador-arm/
22. https://es.wikipedia.org/wiki/Arquitectura_ARM
23. http://www.flytech.es/productos/networking/baja-latencia-infiniband/
24. https://github.com/3dem/relion