Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Automatización de la validación y exactitud de un entorno de virtualización remota de GPUs

Trabajo Fin de Máster

Máster Universitario en Gestión de la Información

Autor: Ismael Bautista Perales

Tutores: Federico Silla Jiménez

Carlos Reaño González

Septiembre 2016

Resumen

En este TFM se va a diseñar e implementar un programa de software que permita realizar de una forma automática tests de validación de software. El objetivo principal es realizar tests masivos del entorno de virtualización de GPUs en clústers. Esto se logrará con el simple hecho de ejecutar un programa en un terminal Linux. Este programa ejecutará una serie de tests de validación y mostrará por pantalla el resultado de los mismos, facilitando en un futuro la acotación y corrección de los errores. Se va a conseguir minimizar los tiempos, ya que ejecutar tests manualmente incrementa mucho tanto el tiempo como los costes derivados del mismo. La automatización de este proceso, además, será aplicable a la validación de otros proyectos software.

Palabras clave: CUDA, rCUDA, NVIDIA, C++, BOOST, virtualización, GPU.

Abstract

This project will design and implement a software program that allows for an automatic validation tests software. The main objective is to conduct mass tests of GPU virtualization environment in clusters. This will be achieved by simply running a program on a Linux terminal. This program will run a series of tests of validation and display on screen the result thereof, facilitating a future dimensioning and correction of errors. It will get minimize the time, since manually run tests greatly increases both the time and the costs thereof. Automating this process also applies to other software validation projects.

Keywords: CUDA, rCUDA, NVIDIA, C++, BOOST, virtualization, GPU.

Tabla de contenidos

1. Introducción 9
2. rCUDA 15
3. Instalación de CUDA y rCUDA 21
4. Mecanismos de testeo, ¿por qué usar la libreria BOOST.Test? 33
5. Evolución del programa: del principio a la versión final 41
6. Experimentos 53
7. Conclusiones 62
8. Bibliografía y referencias 64
9. Anexos 66

1. Introducción

Los titulados de la Escuela Técnica Superior de Ingeniería Informática (ETSINF) de la Universidad Politècnica de València (UPV) reciben una formación muy amplia y diversa durante sus años de estudio. Dicha formación está diseñada para que puedan ejercer con posterioridad su carrera profesional en multitud de áreas relacionadas con la informática. Obviamente, la formación recibida durante los estudios universitarios no puede ser nunca completa y exhaustiva, dada la inmensidad de campos dentro de la informática. Sin embargo, esta formación resulta muy suficiente como un primer y sólido fundamento sobre el que asentar subsiguientes adquisiciones de conocimientos, ya sea a través del estudio de asignaturas propias de las titulaciones de la escuela u otros centros de enseñanza, o bien a través de un proceso autodidacta. Por supuesto, la propia experiencia que dan los años de ejercer una profesión es también inestimable. Además, dada la naturaleza de esta profesión, la cual está en constante evolución, este proceso de adquisición de nuevos conocimientos y experiencia no termina nunca.

Por otra parte, los titulados por la ETSINF de la UPV pueden ejercer su profesión en multitud de áreas diferentes, dado el vasto dominio que es la informática. Es más, a lo largo de su carrera profesional seguramente ejerzan su profesión en puestos de trabajo diferentes en diversas empresas y con atribuciones diferentes, siempre, obviamente, dentro del ámbito de la informática. En este sentido, los titulados de la escuela estarían capacitados, tras completar adecuadamente su formación, para desarrollar tareas, entre otras, como el desarrollo de páginas web, la gestión de proyectos software (y programador en dichos proyectos), administrador de sistemas, desarrollo de soluciones con sistemas empotrados (por ejemplo, Arduino [1], muy utilizado para desarrollar objetos interactivos autónomos), gestor de redes de comunicaciones, creación de nuevos protocolos de comunicación,

participación en la creación de nuevos sistemas hardware tales como procesadores u otros dispositivos del sistema, etc. Esta lista es tan solo una muestra muy pequeña de las posibilidades que se abren a los titulados en informática.

Una de las posibilidades mencionadas anteriormente es la participación en proyectos software. Estos proyectos pueden abarcar multitud de disciplinas y ámbitos. Por ejemplo, se puede crear un proyecto software para el desarrollo de una aplicación. En esta categoría, podríamos encontrar desde aplicaciones para teléfonos móviles hasta aplicaciones científicas capaces de simular cómo se pliega una proteína cuando se encuentra en un disolvente como agua y otros líquidos. Otro proyecto software en el que se puede integrar un titulado en informática sería, por ejemplo, el desarrollo de un sistema operativo. En este sentido podemos encontrarnos desde los clásicos Windows o Linux a los más recientes Android o IOS de Apple. Otros sistemas operativos están destinados a dispositivos más específicos, como automóviles o electrodomésticos (por ejemplo, OSEK/VDX [2]; y JasPar [3].

Siguiendo con los ejemplos de proyectos software en los que un ingeniero en informática podría ejercer su profesión, nos encontramos el software del sistema. En esta categoría podríamos listar software de comunicaciones como la librería MPI ("Message Passing Interface" [4], muy común en centros de computación; o software de virtualización como VMware [5], Xen [6] o VirtualBox [7], muy frecuentes hoy en día en centros de datos. Por ejemplo, este software de virtualización es una componente clave en los servicios cloud ofrecidos por Amazon (Amazon AWS, [8], Google (Google Cloud Platform, [9] o Microsoft (Azure, [10]). Este software de virtualización puede completarse con otras soluciones software como pueden ser los de virtualización de tarjetas gráficas, muy utilizados en soluciones como Citrix [11] o XenServer [12].

En cualquier caso, una característica en común que tienen todos estos proyectos software es que, una vez creado el software, debe ser validado. Este proceso de validación del software debe comprobar que el mismo realiza de una forma correcta las diferentes funciones para las que fue creado y que además su funcionamiento es robusto.

El proceso de validación de cada proyecto software puede ser diferente, dada la diferente naturaleza y propósito de cada proyecto. No obstante, este proceso de validación siempre suele ser costoso de diseñar e implementar, a pesar de que es una pieza fundamental de un proyecto software dado que es este proceso de validación el que garantiza el comportamiento final del producto desarrollado.

En este Trabajo Fin de Máster (TFM) se introduce al alumno en la creación de un proceso de validación, tratando de crear una solución automatizada que sea fácilmente mantenida en el futuro por otros desarrolladores, al tiempo que resulta práctica. En este sentido, se ha creado un entorno de validación automática de software, a través de un programa software desarrollado en el lenguaje C++, utilizando la librería BOOST para realizar los tests que se usarán para validar y un middleware denominado rCUDA, el cual ha sido desarrollado por profesionales de la Universitat Politècnica de València. Este middleware proporciona una virtualización remota de GPUs ("Graphics Processing Units").

En particular, el trabajo desarrollado en este TFM ha sido el siguiente:

- Estudio básico de la librería CUDA, usada para facilitar la programación de las GPUs de NVIDIA.

- Estudio de la librería BOOST, propia del lenguaje C++. Tras su estudio, se ha llevado a cabo una aplicación práctica de la misma con el programa software desarrollado para validar el middleware rCUDA.
- Desarrollo en C++ de la herramienta software de validación del middleware rCUDA.
- Instalación y configuración de un middleware virtualizador de GPUs, tal como es rCUDA.
- Aplicación de técnicas de validación de software de un nivel medio / avanzado.
- Instalación del software de CUDA.

Por otra parte, durante el desarrollo de este TFM el alumno ha adquirido nuevos conocimientos no directamente relacionados con el proceso de validación del software pero que también pueden serle igualmente útiles para el futuro desempeño de su carrera profesional:

- Aprendizaje de comandos y programación en entorno Linux. Asimismo, también de configuración de ficheros y archivos en sistema UNIX.
- Mejora en el uso de la herramienta VIM de Linux, utilizada para el desarrollo del programa. Se contaba con conocimiento previo de la misma debido a su uso por parte del alumno en una etapa profesional previa, y al ser usada en este TFM este conocimiento ha aumentado. Se ha elegido su uso debido a la rapidez y sencillez a la hora de programar el código.
- Aprendizaje y uso de diversas herramientas de compilación de software (GCC, NVCC, MAKE, etc.).

Finalmente, esta memoria se organiza en los siguientes capítulos:

1. Introducción
    - En este capítulo se hace una introducción al TFM, indicando por qué se ha desarrollado y cuales han sido los métodos utilizados en el mismo para ser llevado a cabo.
2. rCUDA
    - En esta parte se presenta la herramienta middleware rCUDA, haciendo hincapié en su arquitectura y su uso. Se exponen también ampliamente ejemplos de evolución de las tarjetas gráficas, así como su aplicación y el uso de su virtualización.
3. Instalación de CUDA y rCUDA
    - En este capítulo se aborda qué es CUDA y se realizan la instalación y configuración de CUDA y rCUDA. También se muestran ejemplos de sus usos.
4. Mecanismos de testeo, ¿por qué usar la librería BOOST.Test?
    - En este capítulo se hace un repaso por diferentes soluciones de testeo de software. Se indican pros y contras de diversas soluciones de testeo, se aborda la librería BOOST y se realiza un pequeño ejemplo en el que se explica de una manera breve y clara un ejemplo de su uso.
5. Evolución del programa: del principio a la versión final
    - En esta parte se aborda la evolución del programa, de tal y como se concibió y diseñó mentalmente en un principio, realizando pruebas y código; hasta una fase intermedia en el que el diseño se asemeja más al de la fase final, el cual es el que se ha acabado imponiendo y utilizando en este TFM. Además, se explican pros y contras de todas las fases, y el por qué se ha ido evolucionando hasta el proyecto final.
6. Experimentos
    - En este capítulo se realizan experimentos con el software programado para este TFM, indicando su funcionamiento tanto

para samples de CUDA, para software de terceros y para el testeo de rCUDA.

7. Conclusiones
    - Conclusiones acerca del TFM y visión de futuro para el software programado.
8. Bibliografía
    - En esta parte del TFM se expone la bibliografía utilizada.
9. Anexo
    - En este capítulo, se encuentra el código fuente del programa, desarrollado en lenguaje C++ usando la librería BOOST.

2. rCUDA

Las máquinas virtuales (VM) han demostrado que proporcionan un ahorro económico para los centros de datos. La razón principal es que varias máquinas virtuales pueden ejecutarse simultáneamente en un único nodo de un clúster compartiendo así sus CPUs. Los costes de adquisición y mantenimiento, por lo tanto, se reducen ya que se requiere una menor cantidad de servidores para hacer frente a la misma carga de trabajo, reduciendo así también las necesidades de consumo de energía. Estos beneficios han provocado que las soluciones de virtualización se hayan vuelto muy populares, dando paso a las soluciones "Cloud" de Amazon, Microsoft o Google; y también han provocado que fabricantes como AMD o Intel incorporen más compatibilidades y mejoras para con la virtualización en el diseño de sus chips.

Las tradicionales tarjetas gráficas que venimos usando en los ordenadores desde hace varias décadas han sufrido una importante evolución con el tiempo. Esta evolución, motivada por un gran volumen de mercado debido al uso masivo de videojuegos, ha permitido que empresas como NVIDIA decidieran realizar una importante inversión económica para mejorar de forma notable sus tarjetas gráficas. El resultado final ha sido que actualmente las tarjetas gráficas tienen una potencia de cómputo tremenda, llegando incluso a los 11 TFlops [13] en una única tarjeta.

Dada su gran potencia computacional, además de usarse para ejecutar videojuegos, las tarjetas gráficas llevan casi una década utilizándose para realizar cómputo de propósito general. Es lo que se conoce como GPGPU ("General Purpose Computing on Graphics Processing Units").

Estas GPU, o aceleradores hardware, también proporcionan grandes beneficios en el contexto de las aplicaciones paralelas, dado que reducen notablemente su tiempo de ejecución, al tiempo que proporcionan importantes ahorros en el consumo de energía dada su excelente relación prestaciones/consumo. El motivo para la gran reducción en el tiempo de ejecución es porque estos aceleradores cuentan con numerosos núcleos que trabajan en paralelo, además de potentes controladores de memoria. Por ejemplo, el modelo Tesla K40 de NVIDIA cuenta con 2880 núcleos y 12 GB de memoria GDDR5 [14] mientras que el modelo Tesla P100 cuenta con 3584 núcleos y 16 GB de memoria GDDR5X [15].

Sin embargo, el uso de GPUs en clústeres de altas prestaciones y centros de datos también tiene ciertas desventajas. Por ejemplo, su alto coste incrementa el coste total del equipamiento del centro de datos. Además, las GPUs requieren una cantidad de espacio nada despreciable, con lo que los costes de espacio en el centro de datos se incrementan al tiempo que la densidad de CPUs se reduce. También, dado que las GPUs consumen una gran cantidad de energía, se hace necesario que los nodos del clúster cuenten con fuentes de alimentación más potentes de lo habitual. Lamentablemente, todos estos incrementos en el coste del hardware no se amortizan posteriormente de forma sencilla porque estos aceleradores suelen presentar una utilización baja, dado que las aplicaciones que los usan no son capaces de mantenerlos ocupados todo el tiempo.

Por otra parte, numerosos usuarios de sistemas Cloud como Amazon AWS necesitan utilizar dichos aceleradores. Sin embargo, conjugar el uso de GPUs con máquinas virtuales es complicado, ya que la GPU debe asignarse en exclusiva a la máquina virtual y no puede ser compartida.

Una forma de solucionar todos estos problemas consiste en la utilización de GPUs virtuales, que son una abstracción de las GPUs reales disponibles en el clúster. Estas GPUs virtuales pueden ser utilizadas desde cualquier nodo del clúster y pueden ser compartidas de forma concurrente por varias aplicaciones. Todo esto hace que el uso de las GPUs se haga mucho más flexible, lo cual se traduce en ahorros de hardware y de energía.

Existen diferentes soluciones de virtualización de GPUs. Algunas de ellas están destinadas a soluciones gráficas, es decir, a virtualizar el escritorio, proporcionando incluso aceleraciones 2D y 3D. Esto es el caso del software de virtualización contenido en VMware, Citrix o XenServer. También en el caso de la GPU de NVIDIA llamada NVIDIA GRID [16]. En el caso de GPGPU, y más concretamente en el caso de CUDA, se han creado recientemente diversas soluciones, con diferentes características. Por ejemplo: V-GPU [17], DS-CUDA [18], GridCuda [19] o la que hemos usado en este TFM, rCUDA [20].

Entre las diversas soluciones de virtualización de GPU destaca rCUDA, por ser la que mejores prestaciones ofrece y también la más moderna con diferencia. rCUDA es un middleware que permite el acceso sin problemas a cualquier dispositivo compatible con CUDA presente en un clúster desde todos los nodos de computación del mismo. Ha sido desarrollado por un equipo de profesionales de la Universidad Politècnica de València (GAP, "Parallel Architectures Group").

rCUDA está estructurado siguiendo una arquitectura cliente - servidor distribuido, tal y como se ve en la figura.

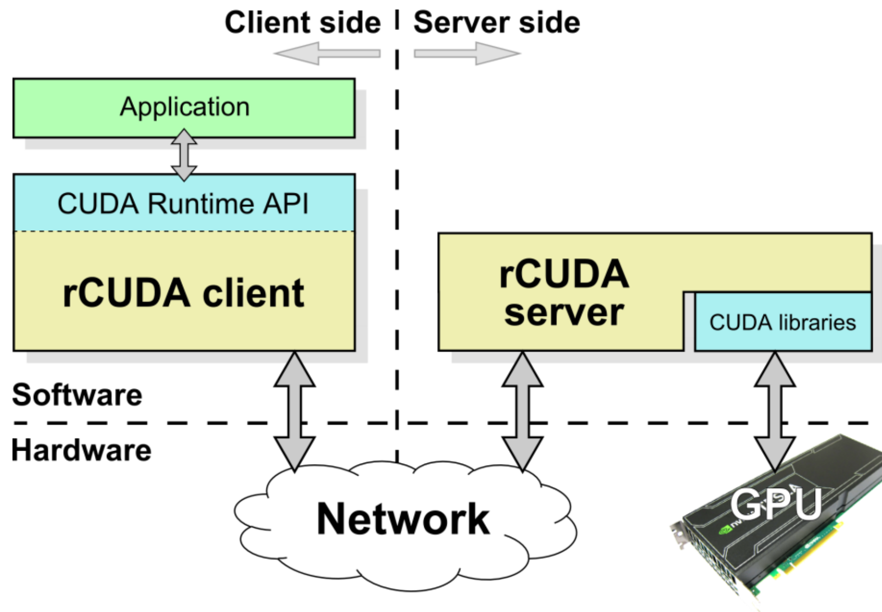

Imagen 1. Arquitectura cliente-servidor de rCUDA.

Las GPUs pueden ser compartidas entre los nodos del clúster, y un único nodo puede utilizar todos estos aceleradores gráficos como si fueran locales. Estas propiedades tienen como objetivo alcanzar unas tasas de utilización del acelerador más altas en el sistema global mientras se reduce simultáneamente la cantidad de recursos necesarios, de espacio y de los requisitos de energía. En rCUDA el cliente middleware se ejecuta en el mismo nodo del clúster que el de la aplicación que solicita los servicios de aceleración GPGPU, proporcionando un reemplazo transparente para las librerías nativas CUDA. El middleware del servidor, por el contrario, se ejecuta en los nodos del clúster desde el que la GPU actual proporciona el servicio GPGPU solicitado.

El cliente rCUDA expone la misma interfaz que la interfaz de NVIDIA CUDA, incluyendo las API de tiempo de ejecución (Runtime API), drivers (Driver API) y librerías CUBLAS, cuFFT, cuRAND y cuSPARSE (solo se excluyen las funciones de CUDA relacionadas con la interoperabilidad gráfica), por lo que las aplicaciones no son conscientes del hecho de que se están ejecutando en la parte superior de una capa de virtualización.

rCUDA incluye soporte para comunicaciones TCP altamente optimizados, y también comunicaciones canalizadas de bajo nivel InfiniBand, así como las características de ser multi-hilo y multi-nodo. Por otra parte, se ha desarrollado una integración de rCUDA con el planificador de trabajos Slurm, permitiendo que los trabajos lanzados a las colas del sistema utilicen GPUs remotas.

rCUDA puede ser útil en tres ámbitos diferentes:

- Clusters: rCUDA permite que una aplicación no MPI haga uso de todas las GPUs del clúster, independientemente del nodo exacto en el que se instalan. Además, rCUDA permite ajustar la cantidad exacta de las GPUs del clúster a las necesidades reales de computación, lo que lleva a una mayor utilización de la GPU y la reducción de los costes generales (energía, adquisición, mantenimiento, espacio, refrigeración, etc.) [21].
- El mundo académico: En el entorno universitario, rCUDA proporciona acceso simultáneo a una GPU de alto rendimiento para muchos estudiantes, lo que reduce los costes de enseñanza [22].
- Máquinas virtuales: rCUDA permite a las aplicaciones que se ejecutan dentro de máquinas virtuales el poder acceder a las GPU instaladas en máquinas físicas remotas, o bien en el propio host que está ejecutando la máquinas virtuales [23] y [24].

Actualmente, rCUDA ha sido probado con éxito con varias aplicaciones seleccionadas de la lista de "Popular GPU-accelerated Applications" de NVIDIA. De esta manera, además de mostrar el comportamiento correcto con los samples de la SDK de NVIDIA (samples que van a ser testeados durante este TFM, junto a otras aplicaciones), rCUDA se ha aplicado a las siguientes aplicaciones: LAMMPS, WideLM, CUDASW ++, HOOMDBlue, mCUDA-MEME, GPU-Blast, Gromacs, GAMESS, DL-Poly, y HPL.

Sin embargo, el desarrollo de un entorno de virtualización remota de GPUs presenta varios problemas. Entre otros, se hace necesario verificar que el entorno virtualizado es correcto y que proporciona los mismos resultados que el entorno sin virtualizar. Precisamente, en este TFM se ha creado un sistema automatizado para la validación del funcionamiento de rCUDA, aunque dicho sistema también podría ser aplicado en otros proyectos software.

3. Instalación de CUDA y rCUDA

Este TFM está realizado bajo la versión 15.07 de rCUDA, la cual soporta a la versión 7 de CUDA y anteriores. El sistema operativo utilizado ha sido Linux Ubuntu 14.04.

Con el objetivo de proporcionar una visión completa del entorno en el que se ha desarrollado este TFM, en este capítulo se revisa la instalación y uso de rCUDA. Dado que rCUDA se apoya en la librería CUDA de NVIDIA, vamos a revisar previamente la instalación y uso de dicha librería.

3.1 Instalación de Cuda

CUDA significa "Compute Unified Device Architecture". El entorno de programación CUDA está formado por un compilador y un conjunto de librerías para el lenguaje C, creadas por NVIDIA. Estas librerías dan la oportunidad de codificar algoritmos para GPUs de NVIDIA. El entorno CUDA incluye también varias herramientas de desarrollo y diversos programas de ejemplo del uso de las librerías CUDA. Estos pequeños programas de ejemplo se conocen comúnmente como "samples". Los samples de CUDA van a ser utilizados en este TFM para realizar los tests de validación del middleware rCUDA.

Primeramente, nos cercioramos de que tenemos una tarjeta gráfica compatible con CUDA. Para ello hay que ejecutar el comando "lspci", el cual lista los dispositivos PCI del sistema. En la imagen 2 se puede observar este proceso:

```
isma@isma-desktop: ~
isma@isma-desktop:~$ lspci | grep -i nvidia
04:00.0 VGA compatible controller: NVIDIA Corporation G96 [GeForce 9400 GT] (rev
 a1)
isma@isma-desktop:~$
```

Imagen 2. Visualización de GPU del equipo.

En este caso, la GPU es una NVIDIA GeForce 9400 GT, la cual es compatible con CUDA de acuerdo con las especificaciones del fabricante.

En segundo lugar, se debe instalar el metapaquete "build-essential", el cual alberga todas las instrucciones para poder programar en C/C++ en un entorno Linux. En una distribución como la que estamos manejando (Ubuntu 14.04) se utilizaría el comando "apt-get", de acuerdo con la imagen 3:

```
isma@isma-desktop: ~
isma@isma-desktop:~$ sudo apt-get install build-essential
[sudo] password for isma:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  cpp-4.8 dpkg-dev fakeroot g++ g++-4.8 gcc-4.8 gcc-4.8-base
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libasan0 libatomic1 libdpkg-perl libfakeroot libgcc-4.8-dev libgomp1 libitm1
  libquadmath0 libstdc++-4.8-dev libstdc++6 libtsan0
Paquetes sugeridos:
  gcc-4.8-locales debian-keyring g++-multilib g++-4.8-multilib gcc-4.8-doc
  libstdc++6-4.8-dbg gcc-4.8-multilib libgcc1-dbg libgomp1-dbg libitm1-dbg
  libatomic1-dbg libasan0-dbg libtsan0-dbg libquadmath0-dbg libstdc++-4.8-doc
Se instalarán los siguientes paquetes NUEVOS:
  build-essential dpkg-dev fakeroot g++ g++-4.8 libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libfakeroot
  libstdc++-4.8-dev
Se actualizarán los siguientes paquetes:
  cpp-4.8 gcc-4.8 gcc-4.8-base libasan0 libatomic1 libdpkg-perl libgcc-4.8-dev
  libgomp1 libitm1 libquadmath0 libstdc++6 libtsan0
12 actualizados, 10 se instalarán, 0 para eliminar y 282 no actualizados.
Se necesita descargar 20,1 MB/32,2 MB de archivos.
Se utilizarán 42,5 MB de espacio de disco adicional después de esta operación.
```

Imagen 3.Instalación del metapaquete "build-essential".

El siguiente paso es instalar CUDA. La última versión de CUDA es la 8.0, y la última versión estable es la 7.5, pero para este TFM se va a instalar la versión 6, ya que es la última versión compatible con la GPU que se va a utilizar para realizar el TFM y las pruebas de validación de rCUDA. Para ello, se debe descargar desde la web de NVIDIA:

http://developer.download.nvidia.com/compute/cuda/6_0/rel/installers/cuda_6.0.37_linux_64.run

Una vez descargada, se extraen los instaladores, tal y como se muestra en la imagen 4:

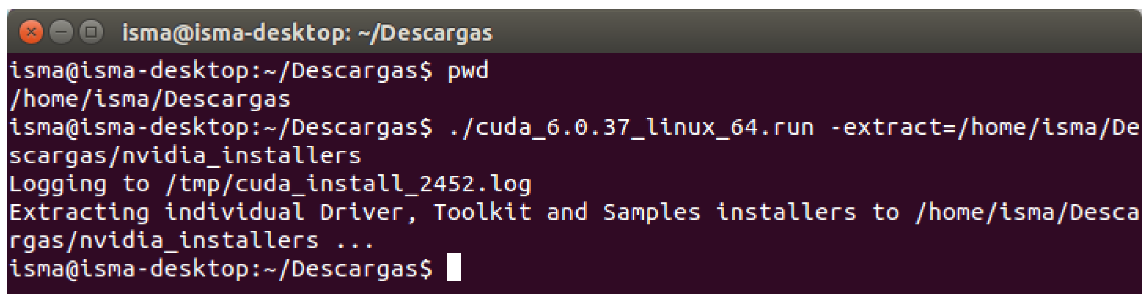

Imagen 4. Extracción de instaladores de Cuda.

Después, se instalan los drivers de NVIDIA mediante la siguiente secuencia de comandos:

```
$ sudo add-apt-repository ppa:xorg-edgers/ppa -y
$ sudo apt-get update

# install the latest version

$ sudo apt-get install nvidia-current
```

Tras la instalación de los drivers es conveniente comprobar que se han instalado correctamente. Para ello el mismo comando "lspci" utilizado previamente, aunque en esta ocasión usaremos la opción "-vnn", la cual hace que se ejecute en modo verbose y se nos muestre tanto la descripción como el número de GPUs de nuestra máquina. Este proceso se puede observar en la imagen 5:

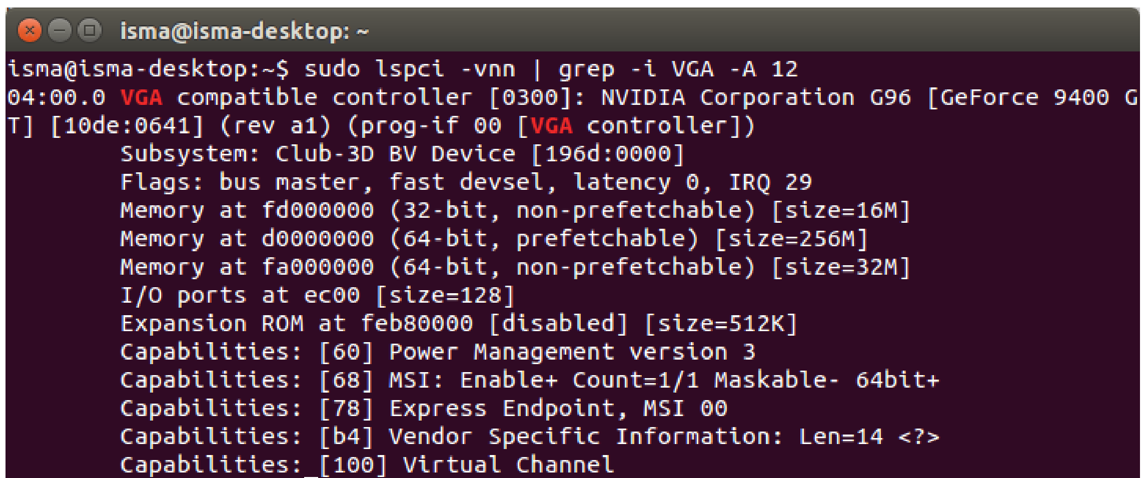

Imagen 5. Comprobación drivers NVIDIA.

Casi finalizando, se debe activar el módulo de NVIDIA, instalar CUDA e instalar los "samples" que contiene CUDA, tal y como se muestra en las imágenes 6 y 7:

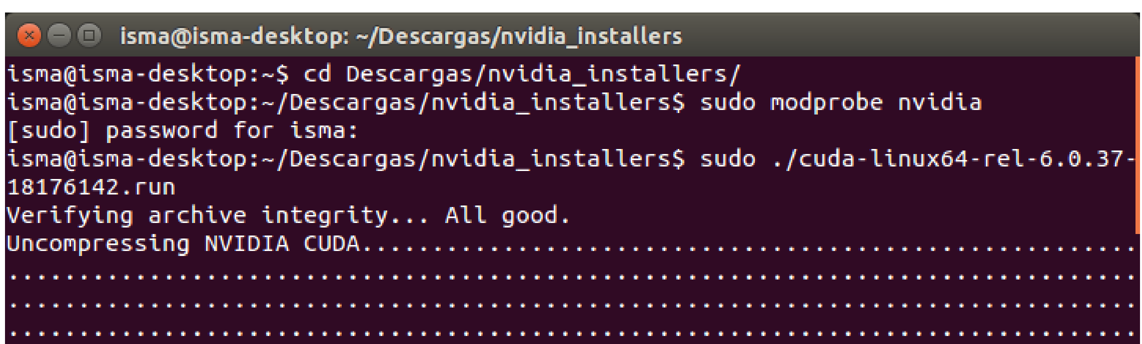

Imagen 6. Activación módulo NVIDIA e instalación de CUDA.

```
isma@isma-desktop: ~/Descargas/nvidia_installers
isma@isma-desktop:~/Descargas/nvidia_installers$ sudo ./cuda-samples-linux-6.0.3
7-18176142.run
Verifying archive integrity... All good.
Uncompressing NVIDIA CUDA Samples.......................................
........................................................................
........................................................................
........................................................................
```

Imagen 7. Instalación de Samples de CUDA.

NVIDIA proporciona una completa guía de instalación de su entorno para facilitar este proceso [25].

3.2 Uso de CUDA

Tras la instalación del entorno CUDA ya lo tenemos listo para su uso. A modo de ejemplo, vamos a ejecutar dos de los samples que incluye el entorno: "deviceQuery" y "bandwidthTest". El sample "deviceQuery" realiza una consulta indicando qué dispositivos GPU tenemos en nuestra máquina, mientras que el sample "bandwidthTest" nos indica el ancho de banda que tiene y soporta nuestra GPU.

El primer paso para ejecutar estos samples es compilarlos. Para ello se deben configurar las variables de entorno $PATH, la cual informa al Shell dónde se encuentran los programas binarios que se pueden ejecutar en el sistema sin necesidad de llamarlos por su ruta absoluta; y $LD_LIBRARY_PATH, que realiza lo mismo que $PATH pero para las librerías. También debe

comprobarse que el compilador de NVIDIA está bien instalado. Este compilador se llama “NVCC”. Este proceso se muestra en la imagen 8:

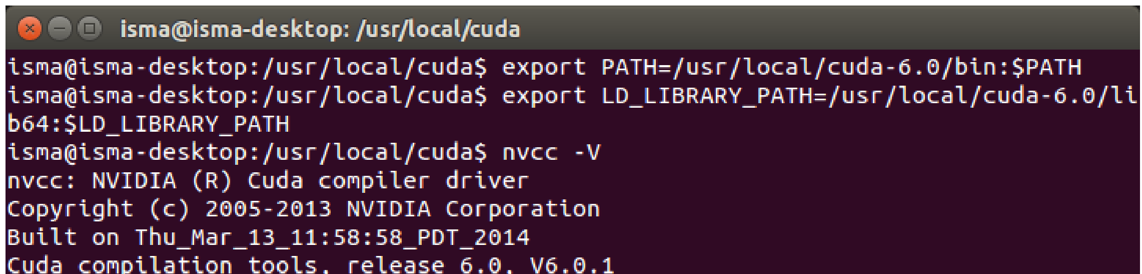

Imagen 8. Comprobación del compilador de NVIDIA “NVCC”.

Tras esta comprobación se procede a compilar los samples. En la imagen 9 se muestra un fragmento de la salida generada por el proceso de compilación:

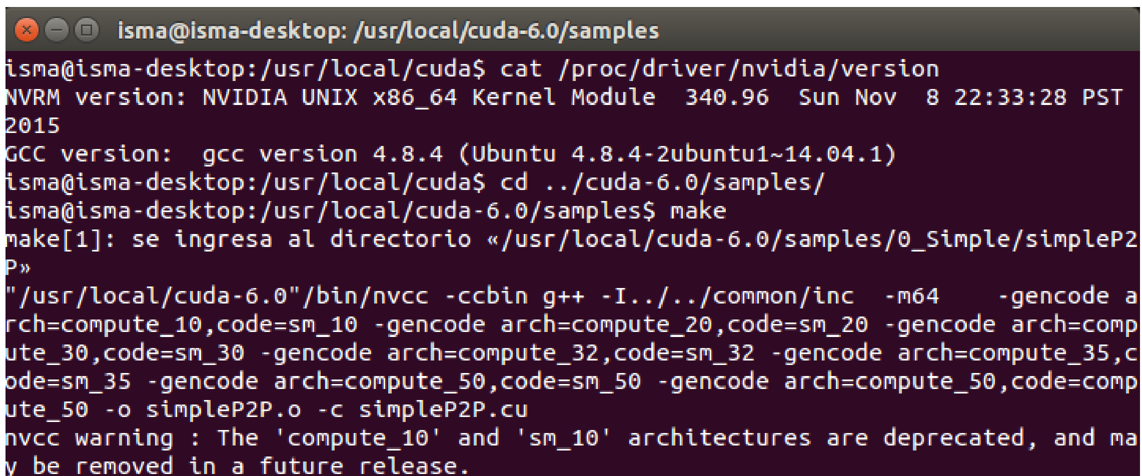

Imagen 9. Compilación de los samples de CUDA.

Una vez completado el proceso de compilación se pueden ejecutar nuestros samples. En la imagen 10 se muestra la salida del sample “bandwidthTest”, el cual nos muestra el ancho de banda de las GPUs sitas en la máquina donde se ejecuta el test:

```
root@isma-desktop:/usr/local/cuda-6.0/samples/bin/x86_64/linux/release# ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce 9400 GT
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     2592.3

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     1969.4

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     7144.1

Result = PASS
```

Imagen 10. Salida por pantalla del test "bandwidthTest".

3.3 Instalación de rCUDA

La instalación de rCUDA es bastante más sencilla e intuitiva que la de CUDA. En primer lugar, se debe solicitar el software a los responsables del proyecto rCUDA en la URL http://www.rcuda.net/index.php/software-request-form.html, donde se puede encontrar el formulario de solicitud. Tras enviar la información requerida, y siempre y cuando el solicitante pertenezca a una institución o empresa, se recibe por correo electrónico un paquete con el software. En el caso de este TFM, se va a usar la versión 15.07, liberada en julio de 2015. Este paquete contiene diversos ficheros binarios y ejecutables, como el demonio de rCUDA; una guía de usuario de rCUDA y las librerías necesarios para su ejecución. Cabe destacar que la descarga y uso de rCUDA no tiene costes.

Para instalar rCUDA, se debe descomprimir el paquete recibido y moverlo a la carpeta deseada. En el caso de este TFM, se ha optado por mover los

archivos a la ruta “/usr/local”, tal y como se muestra en las imágenes 11 y 12. Con esto se completa la instalación de rCUDA:

```
root@isma-desktop:/home/isma/Descargas# tar -xzvf rCUDAv15.07-CUDA6.0-Ubuntu-14.04.tgz
rCUDAv15.07-CUDA6.0/
rCUDAv15.07-CUDA6.0/contents.txt
rCUDAv15.07-CUDA6.0/lib/
rCUDAv15.07-CUDA6.0/lib/libcublas.so.6.0
rCUDAv15.07-CUDA6.0/lib/libcudnn.so.6.5
rCUDAv15.07-CUDA6.0/lib/libnppc.so.6.0
rCUDAv15.07-CUDA6.0/lib/libcuda.so
rCUDAv15.07-CUDA6.0/lib/libcublas.so
rCUDAv15.07-CUDA6.0/lib/libcurand.so
rCUDAv15.07-CUDA6.0/lib/libcuda.so.1
rCUDAv15.07-CUDA6.0/lib/libcurand.so.6.0
rCUDAv15.07-CUDA6.0/lib/libcudnn.so
rCUDAv15.07-CUDA6.0/lib/libnppi.so.6.0
rCUDAv15.07-CUDA6.0/lib/libcuda.so.346.46
rCUDAv15.07-CUDA6.0/lib/libcudart.so.6
rCUDAv15.07-CUDA6.0/lib/libcudnn.so.6
rCUDAv15.07-CUDA6.0/lib/comm/
rCUDAv15.07-CUDA6.0/lib/comm/rCUDAcommTCP.so
rCUDAv15.07-CUDA6.0/lib/comm/rCUDAcommIB.so
rCUDAv15.07-CUDA6.0/lib/libcusparse.so.6.0
rCUDAv15.07-CUDA6.0/lib/libcusparse.so.6
rCUDAv15.07-CUDA6.0/lib/rCUDAcommTCP.so
rCUDAv15.07-CUDA6.0/lib/libcufft.so.6
rCUDAv15.07-CUDA6.0/lib/libcusparse.so
```

Imagen 11. Descompresión del paquete de rCUDA

```
root@isma-desktop: /usr/local
root@isma-desktop:/home/isma/Descargas# mv rCUDAv15.07-CUDA6.0 /usr/local/
root@isma-desktop:/home/isma/Descargas# cd /usr/local/
root@isma-desktop:/usr/local# ls
bin  cuda  cuda-6.0  etc  games  include  lib  man  rCUDAv15.07-CUDA6.0  sbin  s
```

Imagen 12. Copia de rCUDA al directorio “usr/local”.

3.4 Uso de rCUDA

Con el fin de ilustrar el uso de rCUDA, vamos a ejecutar los mismos programas de prueba que se han ejecutado en la sección 3.2.

Cabe destacar que el entorno rCUDA es transparente a las aplicaciones, es decir, éstas no son conscientes de que están usando una GPU remota. De hecho, dado que rCUDA es compatible binario con CUDA, no es necesario siquiera recompilar las aplicaciones CUDA para que puedan utilizar rCUDA, es decir, el mismo binario utilizado con la librería CUDA se puede utilizar

con rCUDA, siempre y cuando hubiera sido compilado para utilizar librerías dinámicas.

En el capítulo 2 se ha explicado que rCUDA presenta una arquitectura cliente-servidor distribuida. En este sentido, en el ordenador del clúster donde se encuentra la GPU se ejecuta el servidor rCUDA mientras que la parte cliente de rCUDA se ejecuta en el mismo ordenador donde se ejecuta la aplicación acelerada. De este modo, dado que con rCUDA hay dos actores, es necesario configurar adecuadamente cada uno de ellos.

Primeramente, vamos a ejecutar el demonio de rCUDA (el servidor). Para ello, se deben exportar las variables de entorno necesarias y posteriormente ejecutar el demonio con los siguientes comandos. El primero exporta la variable LD_LIBRARY_PATH la librería de CUDA necesaria y el segundo comando ejecuta el demonio de rCUDA en modo verbose y loopback, esto es, utilizando nuestro propio ordenador como servidor y cliente. Se ha decidido usar una máquina única debido a que facilitaba el trabajo a la hora de realizar el desarrollo del código y los tests, ya que no se dependía de conexiones externas:

```
export LD_LIBRARY_PATH=/usr/local/cuda-
6.0/lib64:$LD_LIBRARY_PATH

./rCUDAd -ilv
```

En la parte cliente, y antes de ejecutar la aplicación, se deben exportar las variables de entorno correspondientes. Los samples que hemos compilado en la sección 3.2 no fueron compilados para utilizar librerías dinámicas. Debemos recompilarlos para que hagan uso de este tipo de librerías. Para ello utilizamos el comando siguiente, el cual permite utilizar este tipo de librerías:

```
make EXTRA_NVCCFLAGS=--cudart=shared
```

Tras esto, debemos exportar las variables de entorno correspondientes. Al usar el modo "loopback", indicamos que el "Device 0" (nuestra GPU) se encuentra en la IP 127.0.0.1:

```
export LD_LIBRARY_PATH=/usr/local/rCUDAv15.07-
CUDA6.0/bin/:$LD_LIBRARY_PATH
```

```
export RCUDA_DEVICE_0="127.0.0.1"
export RCUDA_DEVICE_COUNT="1"
```

Se van a mostrar dos ejemplos de ejecución. En el primero, el sample "bandwidthTest", cuando se ejecuta con rCUDA, proporciona la siguiente salida:

```
./bandwidthTest
```

```
root@isma-desktop:/usr/local/cuda-6.0/samples/1_Utilities/bandwidthTest# ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce 9400 GT
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     2592.1

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     1950.2

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     7146.6

Result = PASS
```

Imagen 13. Ejecución de bandwidthTest.

Finalmente, al acabar el test, el verbose del demonio de rCUDA nos devuelve esta salida, lo que denota qué se está ejecutando correctamente, ya que está recibiendo información:

```
root@isma-desktop: /usr/local/rCUDAv15.07-CUDA6.0/bin
rCUDAd[2721]: 'cudaMalloc' received.
rCUDAd[2721]: 'cudaMalloc' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaEventRecord' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaMemcpy' received.
rCUDAd[2721]: 'cudaEventRecord' received.
rCUDAd[2721]: 'cudaDeviceSynchronize' received.
rCUDAd[2721]: 'cudaEventElapsedTime' received.
rCUDAd[2721]: 'cudaEventDestroy' received.
rCUDAd[2721]: 'cudaEventDestroy' received.
rCUDAd[2721]: 'cudaFree' received.
rCUDAd[2721]: 'cudaFree' received.
rCUDAd[2721]: 'cudaDeviceReset' received.
rCUDAd[2721]: Remote application finished.
```

Imagen 14. Salida por pantalla del demonio de rCUDA tras ejecución de bandwidthTest.

La segunda prueba se realiza con el sample “deviceQuery”. En él se muestran las especificaciones de la GPU sita en la máquina donde se ejecuta el programa:

```
./deviceQuery Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "GeForce 9400 GT"
  CUDA Driver Version / Runtime Version          6.5 / 6.0
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 1023 MBytes (1073020928 bytes)
  ( 2) Multiprocessors, (  8) CUDA Cores/MP:     16 CUDA Cores
  GPU Clock rate:                                1350 MHz (1.35 GHz)
  Memory Clock rate:                             400 Mhz
  Memory Bus Width:                              128-bit
  Maximum Texture Dimension Size (x,y,z)         1D=(8192), 2D=(65536, 32768), 3D=(2048, 2048, 2048)
  Maximum Layered 1D Texture Size, (num) layers  1D=(8192), 512 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(8192, 8192), 512 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  768
  Maximum number of threads per block:           512
  Max dimension size of a thread block (x,y,z): (512, 512, 64)
  Max dimension size of a grid size    (x,y,z): (65535, 65535, 1)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             256 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      No
  Device PCI Bus ID / PCI location ID:           4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.5, CUDA Runtime Version = 6.0, NumDevs = 1, Device0 = GeForce 9400 GT
Result = PASS
```

Imagen 15. Ejecución de deviceQuery.

Tal y como sucede en el ejemplo anterior, el demonio de rCUDA nos devuelve esta salida, de la cual se denota que se está ejecutando correctamente ya que está recibiendo información:

```
root@isma-desktop:/usr/local/rCUDAv15.07-CUDA6.0/bin# ./rCUDAd -ilv
rCUDAd v15.07
Copyright 2009-2015 UNIVERSITAT POLITECNICA DE VALENCIA. All rights reserved.
rCUDAd[3797]: Using rCUDAcommTCP.so communications library.
rCUDAd[3797]: Server daemon succesfully started.
rCUDAd[3800]: Trying device 0...
rCUDAd[3800]: CUDA initialized on device 0.
rCUDAd[3800]: Connection established with localhost.
rCUDAd[3800]: 'cudaGetDeviceProperties' received.
rCUDAd[3805]: Trying device 0...
rCUDAd[3800]: 'cudaDriverGetVersion' received.
rCUDAd[3800]: 'cudaGetDeviceProperties' received.
rCUDAd[3805]: CUDA initialized on device 0.
rCUDAd[3800]: 'cudaDeviceReset' received.
rCUDAd[3800]: Remote application finished.
```

Imagen 16. Salida por pantalla del demonio de rCUDA tras ejecución de deviceQuery.

4. Mecanismos de testeo, ¿por qué usar la librería BOOST.Test?

Hoy en día el testeo de software se realiza en todos los desarrollos profesionales de código de la industria: no hay software no testeado que llegue a manos de un potencial cliente comprador.

En primer lugar, y por definición [26] se plantea una prueba de software como que "la verificación se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica. La validación se refiere a un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente".

El programa software diseñado y programado en este TFM se encarga de realizar tests unitarios. Un test unitario es una forma, basada en código, de comprobar el funcionamiento correcto de un módulo de código. Los tests unitarios nos proporcionan la certeza de que cada uno de los módulos que forman una unidad de código funcionan correctamente individualmente, esto es, por separado. Las ventajas de realizar tests unitarios de software son muchas, como por ejemplo:

- Fácil localización de errores.
- Se detectan los errores en etapas de desarrollo, por lo que se previenen errores en tiempo de ejecución.
- Se acotan los errores, ya que al dividir por módulos los tests, es mucho más sencillo saber qué está fallando y en qué condiciones.
- Se consigue que haya menos errores al eliminar errores previos, ya que al corregir fallos anteriores se debe volver a lanzar el test, y observar de una manera mucho más sencilla si falla algo más por qué es.

- Nos aporta una potente herramienta que comprueba que no se han introducido errores tras añadir una nueva funcionalidad o bien intentar arreglar un error previo. Esto reduce el coste de mantenimiento del código
- Permite refactorizar el código en un futuro con cierta seguridad.
- Documentan el código. Los resultados de los tests sirven como descripción del funcionamiento de las funciones.
- Se acelera el desarrollo del software.

Actualmente, debido al auge del uso de esta metodología, hay muchas y variadas herramientas que se utilizan para el testeo de software, entre las que destacan estas tres:

- Frameworks.
- Plug-ins bajo el manto de un navegador web.
- Librerías.

4.1 Estructura del proceso de validación para rCUDA

Desde en un principio, en este TFM se han ido barajando diferentes alternativas a la hora de diseñar y codificar la estructura del proceso de validación para rCUDA. De hecho, la evolución del pensamiento para con esta estructura ha ido variando a lo largo del desarrollo del trabajo fin de máster.

En un inicio se pensó en diseñar tests unitarios dentro de cada uno de los samples que ofrecía CUDA, para así validar rCUDA. Esta forma de plantear la validación llevaba a una subida en la escala de tiempo considerable, ya que habría que haber modificado más de un centenar de ficheros para poder tener todos los tests operativos. Además, este proceso de validación provocaba que hubiera que ejecutar los samples uno a uno, por lo que su

calidad y utilidad hubiera sido bastante pobre. Finalmente, haber utilizado este método hubiera provocado que cada vez que hubiera salido una nueva versión de CUDA, se hubiera tenido que revisar cada uno de los ficheros ejecutables donde se habría codificado el test para poder adaptarlos a los nuevos cambios.

Tras esto, se planteó la idea de generar un único programa, dividido en su interior en diferentes módulos de tests, que apuntaran a directorios fijados manualmente dentro del mismo código. Esto mejoraba mucho, respecto al método explicado en el párrafo anterior, los tiempos de ejecución de los tests; y mejoraba ligeramente el mantenimiento del futuro código. Pero se presentaba un problema: el programa debería estar en una ruta obligatoriamente, para que a la hora buscar las rutas y ejecutar los tests se encontraran los archivos de CUDA. Además, planteaba dificultades para un futuro mantenimiento, ya que en versiones siguientes de CUDA aparecerán nuevos samples, otros desaparecerán y se puede modificar la estructura de los directorios, lo que llevaría a este software a su no utilidad.

La estructura que se ha llevado a cabo finalmente ha sido la siguiente. En primer lugar, se ha creado un único programa que realiza la validación de rCUDA desde una ruta indicada en la ejecución del mismo, descendiendo en los directorios inferiores buscando archivos ejecutables y ficheros de configuración que indican cómo ejecutar estos archivos en cada uno de los casos según lo escrito en su interior, indicando por la salida estándar (por la pantalla) qué tests han sido satisfactorios y qué tests han resultado erróneos. La ventaja de usar esta estructura radica en que el código es muy fácil de mantener; está concentrado en un punto, lo que lo hace mucho más accesible; permite que en futuras versiones de CUDA, cuando se quiera testear rCUDA, con el simple hecho de indicar en un fichero de configuración una nueva ruta indicando como se quiere ejecutar, se realizará el test; que se

puede ejecutar desde cualquier ruta; etc. En el siguiente capítulo del TFM se aborda con más profundidad los detalles de la estructura elegida.

4.2 Cómo implementar el mecanismo de validación

Existen infinidad de frameworks, los cuales facilitan los tests unitarios de software: en este TFM se van a nombrar de una manera breve algunos frameworks de testeo; y en el siguiente capítulo se va a dar a conocer la librería de C++ "BOOST", la cual ha sido la elegida para realizar los tests en este trabajo, de una manera más amplia.

Lista de otros frameworks de C++ útiles para el testeo de software

- CppUnit: en mi opinión, junto a BOOST y CxxTest (de la cual hablo más abajo) es el framework más completo. Es portable, con mucha documentación, y pese a que en sus inicios era compleja de usar, actualmente con muy pocas líneas de código se pueden conseguir resultados muy satisfactorios. En este enlace se puede ver un ejemplo de testeo con CppUnit: http://cppunit.sourceforge.net/doc/cvs/money_example.html.
- NanoCppUnit: framework con escasa bibliografía y solo válido en entornos Windows. No se ha encontrado un ejemplo que destile cierta relevancia o interés.
- Unit++: framework con escasa bibliografía. Se basa mucho en C++ para su implementación interna. En este enlace se puede ver un ejemplo de testeo con Unit++: http://unitpp.sourceforge.net/first.html
- CxxTest: este framework goza de bastante bibliografía. Utiliza el lenguaje Perl para generar código C++, por lo que uno de sus requisitos es instalar y configurar Perl de una forma correcta. Es fácilmente portable y con una pequeña cantidad de código se consiguen resultados muy buenos. En este enlace se puede ver un ejemplo, además de una guía de uso, de CxxTest:

http://cxxtest.com/guide.html

En este enlace de Wikipedia se puede observar una tabla, en la cual se comparan infinidad de frameworks de C++:

https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C.2B.2B

Finalmente, se ha decidido utilizar la librería BOOST, debido a que su documentación me ha parecido más entendible y con más ejemplos prácticos unidos al lenguaje C++, lo cual ayuda a la hora de elaborar y utilizar el potencial que ofrece BOOST. Tal y como se ha comentado en los ejemplos del párrafo anterior, CxxTest es el que más cerca estaría en prestaciones, en mi opinión, de BOOST, pero sin embargo depende de otro lenguaje de programación más (Perl) el cual tendría que interpretar y aprender algo, por lo que esto acabó de alentar a que este TFM se basara en BOOST. Tras este también era interesante CppUnit, pero se decidió elegir BOOST debido a que, en mi opinión, su uso es ligeramente más intuitivo y más sencillo. En pocas palabras, BOOST necesita menos líneas de código para realizar lo mismo que CppUnit.

4.3 BOOST

BOOST, además de ser un framework de testeo de software, sirve para otras muchas finalidades. La librería BOOST.Test, que es la que se utiliza en este TFM, tiene un gran potencial y soporte para el manejo de errores y excepciones.

- Se necesita una cantidad muy pequeña de código para conseguir unos tests muy fiables, intuitivos y por lo tanto muy útiles.
- El código es sencillo de modificar y muy fácil de portabilizar.
- Su bibliografía y comunidad de apoyo en Internet es muy amplia: hay mucha cantidad de manuales con referencias a funcionalidades, tanto

en la web de http://boost.org, como en otras de ámbito privado o en la misma GitHub.

- Se encuentran una gran cantidad de macros con una gran cantidad de usos: facilita mucho la captura de errores.

Como todos los frameworks, y todas las librerías, es muy importante invertir unos días en estudiar el funcionamiento básico del mismo, para así poder sacar más partido de las facilidades que éste nos ofrece. Sin un tiempo de preparación previo, un framework no resulta de utilidad ya que no es posible conocer su funcionamiento trivialmente.

Para instalarlo, se debe descargar la librería del siguiente enlace:

https://sourceforge.net/projects/boost/files/boost/1.60.0/boost_1_60_0.tar.gz/download

Tras descomprimir los archivos y copiarlos en la ruta deseada, se debe compilar con los siguientes comandos, y ya estará instalada en el equipo:

```
./bootstrap.sh -prefix=/usr/local/
./b2 install
```

La instalación de BOOST es bastante sencilla, y su curva de aprendizaje al principio es muy corta, por lo que en seguida se tienen nociones para poder realizar cosas interesantes. Si se quiere tener un dominio amplio de la librería BOOST.test sí que se requiere, tras el primer periodo de aprendizaje algo más escueto, una cantidad de tiempo bastante grande debido a la gran cantidad de opciones y variables que nos ofrece la librería.

Ejemplo práctico del uso de BOOST

En el ejemplo de la imagen inferior [27], se realiza un testeo organizado en “suites”. Estas “suites” (BOOST_AUTO_TEST_SUITE) están creadas para delimitar los tests, esto es, para cuando hay un número muy elevado de los mismos, poder organizarlos en zonas. De esta manera, cuando se ejecutan

los tests y se muestran los resultados por pantalla, es más fácil situar donde está el test debido a que por la salida estándar se muestra cuando se inicia y cuando se acaba una "suite". Estas "suites" están englobadas en zonas de testeo definidas en un nivel superior por módulos (BOOST_TEST_MODULE). Dentro de cada "suite" se declaran los diferentes tests, usando la macro BOOST_AUTO_TEST_CASE. Dentro de estas macros, se incluyen las acciones que se requiere que se realicen por parte de los tests. Por ejemplo, en el caso del test "universeInOrder" el test comprueba que la suma de los dos números enteros sea igual a cuatro: si no lo fuera, esto es, si se le pasara por parámetro dos números cuya suma no fuera cuatro, al ejecutar el test de ese módulo aparecería un error de ejecución en la salida estándar.

```
1   #define BOOST_TEST_DYN_LINK
2   #define BOOST_TEST_MODULE Suites
3   #include <boost/test/unit_test.hpp>
4
5   int add(int i, int j)
6   {
7       return i + j;
8   }
9
10  BOOST_AUTO_TEST_SUITE(Maths)
11
12  BOOST_AUTO_TEST_CASE(universeInOrder)
13  {
14      BOOST_CHECK(add(2, 2) == 4);
15  }
16
17  BOOST_AUTO_TEST_SUITE_END()
18
19  BOOST_AUTO_TEST_SUITE(Physics)
20
21  BOOST_AUTO_TEST_CASE(specialTheory)
22  {
23      int e = 32;
24      int m = 2;
25      int c = 4;
26
27      BOOST_CHECK(e == m * c * c);
28  }
29
30  BOOST_AUTO_TEST_SUITE_END()
```

Imagen 17. Ejemplo de uso de BOOST con Suites.

Como nota, indicar que cuando se ejecuta un test, se ejecutan todas las "suites" que lo conforman.

En la ejecución del testeo de rCUDA en este TFM se ha seguido un patrón parecido al de este pequeño trozo de código, pero con una complejidad mayor y realizando las llamadas a la librería BOOST con una notación algo distinta.

5. Evolución del programa: del principio a la versión final

El objetivo de un programa de validación es verificar que el software que va a analizar cumple con las especificaciones recibidas y que realiza el propósito para el que fue creado de una manera correcta. En este TFM, el objetivo del programa realizado es validar el middleware rCUDA, basándose en la salida estándar que produce en test para saber si se ha producido un error o no. Asimismo, este programa, además de servir para usarse con samples CUDA, también está diseñado para ser utilizado para testear software de otros ámbitos, tal y como se demuestra y argumenta más ampliamente en el siguiente capítulo, lo que lo convierte en un software con una gran cantidad de usos dentro de lo que es testeo de programas.

Debido a su diseño, el programa realizado en este TFM es fácilmente mantenible y ampliable, ya que está claramente delimitado por funciones, tal y como se puede observar en el Anexo de este TFM, además de estar comentado de una manera exhaustiva para facilitar la comprensión del código para futuros desarrolladores. Además, este software se adapta fácilmente a nuevas versiones de CUDA, debido a que su funcionamiento usando métodos recursivos y ficheros de configuración para buscar los ficheros ejecutables consigue que aunque se añadan nuevos samples a versiones futuras de CUDA, con simplemente añadir un fichero de configuración si fuera necesario (si no se añade, se ejecutará el test sin parámetros) se conseguiría testear su funcionamiento. En el caso de que en futuras versiones se eliminaran tests que existen en versiones actuales, no afectaría al programa, debido a que no encontraría la ruta al ejecutarse el programa de validación, por lo que para el programa de este TFM ese test ya no existiría.

A continuación vamos a mostrar la evolución del diseño del programa. Mostramos tres pasos de esta evolución: la primera versión realizada, una versión intermedia y finalmente la versión definitiva.

5.1 Versión inicial

En un principio, en este TFM se pensó en utilizar la librería BOOST dentro de cada uno de los samples de CUDA, utilizando Macros que englobaran el código y así ejecutar los tests.

Al poco tiempo de intentar programar así, se llegó a la conclusión de que era inviable ya que, además de todos los problemas que estaba dando el programa a la hora de compilar librerías de CUDA, de C++ y de BOOST a la vez, hubiera habido que modificar centenares de ficheros, lo cual habría provocado lo siguiente:

- Imposibilidad de mantenimiento futuro: el programa estaría dispersado en tal cantidad de ficheros que sería imposible su mantenimiento. Ante cualquier cambio de versión de CUDA, el software de testing quedaría inhabilitado ya que habría que reprogramar todos los samples de nuevo.
- Software no útil: con la realización de este programa lo que se quiere es reducir tiempos: si se realizar de esta manera, la inversión en tiempo se multiplicaría exponencialmente, lo cual quitaría sentido al TFM, el cual busca el decremento tiempos de ejecución y testing.
- Software no adaptable: el mecanismo de testing solo serviría para los samples de CUDA, sin capacidad de configurar nada ni de ser usado para otro tipo de tests, debido a que se programaría directamente sobre el software de CUDA, y no externamente.

En la imagen inferior se muestra un pequeño ejemplo de una parte de código de BOOST inmerso en mitad del sample “matrixMul”. Hay que tener en cuenta que hay samples con centenares de líneas de código, por lo que haber programado de esta manera, buscando dónde realizar los tests y cómo crear un código robusto para con ellos, y multiplicar esto por centenares de tests, evidencian que esta opción no era una buena forma de gestionar este programa:

```
    if (correct)
    {
        //return EXIT_SUCCESS; //IBP - Quitar esto y test de Boost
        matrix_res2(correct);
     }
    else
    {
        return EXIT_FAILURE;
    }
}



int matrix_res(int mr){
        return mr;
}

BOOST_AUTO_TEST_CASE ( my_test ){
        int a = (long)matrix_res;
        BOOST_CHECK(a > 0);
}


/**
 * Program main
 */
int main(int argc, char **argv)
{
    printf("[Matrix Multiply Using CUDA] - Starting...\n");

    if (checkCmdLineFlag(argc, (const char **)argv, "help") ||
        checkCmdLineFlag(argc, (const char **)argv, "?"))
    {
        printf("Usage -device=n (n >= 0 for deviceID)\n");
        printf("      -wA=WidthA -hA=HeightA (Width x Height of Matrix A)\n");
        printf("      -wB=WidthB -hB=HeightB (Width x Height of Matrix B)\n");
        printf("  Note: Outer matrix dimensions of A & B matrices must be equal.\n");

        exit(EXIT_SUCCESS);
```

Imagen 18. Ejemplo de la primera versión del programa. Sample “matrixMul”.

5.2 Versión intermedia

Tras ver que esta solución no era viable, se decidió realizar un único programa en C++, el cual utilizara las macros de BOOST para realizar los tests unitarios y ejecutara la salida del programa por la salida estándar. Las macros utilizadas eran: "BOOST_AUTO_TEST_SUITE", para delimitar los grupos de tests; y "BOOST_AUTO_TEST_CASE", las cuales englobaban cada uno de los tests de CUDA. Esta versión evolucionada aportaba más sencillez y velocidad a la hora de programar, ya que en un solo fichero se podía tener las llamadas al testing de todos los samples, pero seguía teniendo problemas graves:

- Para ejecutar los tests de CUDA, había que "hardcodear" las rutas de los ejecutables de los ficheros, una por una, en el código del programa. Esto es una práctica de programación nada aconsejable ya que elimina la propiedad de atomicidad de la misma.
- Hay tests que necesitan ser ejecutados con parámetros: habría que "hardcodear" estos tests, individualmente, dentro del código.
- En futuras versiones de CUDA, en cuanto se cambiara la ruta de algún ejecutable de un directorio, habría que modificar el código de ejecución de este test a mano. Además, los nuevos tests que surgieran habría que programarlos creando un nuevo "BOOST_TEST_CASE" explícito para ello. De todo esto se concluye que el mantenimiento del programa sería costoso.

En la imagen inferior se muestra un pequeño ejemplo de cómo hubiera sido el patrón a seguir:

```
#include <stdio.h> // Quitar en version final
#include <stdlib.h>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Hello
#include <boost/test/unit_test.hpp>
#include <sys/types.h>
#include <dirent.h>

BOOST_AUTO_TEST_SUITE(Simple)

BOOST_AUTO_TEST_CASE(matrixMul)
{
    //printf("Output %d.", system("./matrixMul  -wA=0"));
    BOOST_CHECK(system("./0_Simple/matrixMul/matrixMul > /dev/null")==0);
}

BOOST_AUTO_TEST_CASE(vectorAdd)
{
    BOOST_CHECK(system("./0_Simple/vectorAdd/vectorAdd > /dev/null")==0);
}

BOOST_AUTO_TEST_SUITE_END()

BOOST_AUTO_TEST_SUITE(UTILITIES)

BOOST_AUTO_TEST_CASE(bandWidthTest)
{
        BOOST_CHECK(system("./1_Utilities/bandwidthTest/bandwidthTest > /dev/null")==0);
}

BOOST_AUTO_TEST_SUITE_END()
```

Imagen 19. Segunda evolución en la programación del software. Ejemplo.

5.2 Versión final

Finalmente, se decidió realizar un único programa en C++, pero con una serie de características que le dan muchísima robustez y facilidad de mantenimiento:

- Programa basado en recursividad: recorre todos los directorios en profundidad a partir del directorio que se le proporciona por parámetro en la ejecución del programa. Así se evita tener que "hardcodear" las rutas, ya que el programa busca todos los tests ejecutables en todos los directorios en los que entra, y si existe, lo ejecuta. En la imagen inferior se muestra este funcionamiento: la función "init_unit_test_suite" ejerce de función main del programa, a la vez que crea una suite de BOOST. Este método main añade a esta suite un test de BOOST, llamando a una función denominada "cuentaArchivos". Esta función a su vez llama recursivamente a la función "cuentaArchivos2", cuantas veces se vaya metiendo en

diferentes directorios, función en la cual se realiza la búsqueda de directorios, se realizan los tests y comprobaciones de ficheros de configuración, etc.

```
test_suite* init_unit_test_suite( int argc, char* argv[] )
{
  unsigned num;

  if (argc != 2)
    {
      error("Uso: ./directorio_2 <ruta>\n");
    }
  printf("\nEjecutando tests en la ruta: %s\n\n", argv[1]);
  ruta_main=argv[1];
  niv_main=1;
  framework::master_test_suite().add( BOOST_TEST_CASE( &cuentaArchivos ) );

  return EXIT_SUCCESS;

}

void cuentaArchivos()
{
    cuentaArchivos2(ruta_main, niv_main);
}

unsigned cuentaArchivos2(char *ruta, int niv)
{
    /* Con un puntero a DIR abriremos el directorio */
    DIR *dir;
    /* en *ent habrá información sobre el archivo que se está "sacando" a cada momento */
    //El programa continúa, realizando los tests, comprobaciones, etc.
```

Imagen 20. Funcionamiento de la recursividad en el programa.

- Este software permite usar los mecanismos de test tanto para los samples de CUDA como para tests de otros ámbitos: se mejora de una manera muy notable la usabilidad del programa en el futuro. En puntos siguientes se abordarán diferentes experimentos de funcionamiento del software.
- En esta versión se tomó la idea de la creación de archivo MAKEFILE en la ruta, para facilitar la compilación del programa. Con solo hacer “make” en la ruta donde está, se ejecutan todos comandos de compilación de C++ necesarios para este archivo. Esto está orientado para futuros mantenimientos del programa, facilitando las labores de desarrollo y compilación. Cabe decir que en la versión anterior también se hubiera podido utilizar.

- Utilización de fichero de configuración: se ha diseñado el programa para que busque en cada uno de los directorios donde va a ejecutar un test un fichero de configuración “test.config”. Si lo encuentra, lee su contenido y, dependiendo de sus valores, ejecuta el test de una de las siguientes maneras:
    - Si el valor es 1, el test se ejecuta, y si además tiene argumentos ejecuta los argumentos también. Las líneas que empiezan por '#' (comentarios) son ignorados.
    - Si el valor es 2, el test no se ejecuta y se muestra mensaje al usuario de que no se va a ejecutar.
    - Si no hay fichero de configuración, se ejecuta el test sin argumentos.

En la imagen inferior se puede observar el comportamiento del programa en la búsqueda del fichero de configuración. La explicación del funcionamiento está explicada en los comentarios del código sitos en la imagen. En resumen, lo que realiza esta parte de código es buscar un fichero de configuración en la ruta donde está el test a ejecutar, y según sea el resultado, ejecuta el test ya sea con o sin argumentos, o bien ignora el test si el fichero de configuración así lo ordena:

```
    printf("%sEjecutando test: %s\n", posstr, nombrecompleto);
    comandocompleto=getFullCommand(ruta, ent); //Sin argumetnos
comandocompleto_argumentos=getFullCommand_ArgumentsCase(ruta, ent); //Con argumentos

bool ejecutado = false; //Para comprobar si ha entrado en a=1, no haga una segunda ejecución del test
bool con_argumento = false;
bool sin_argumento = false;
    std::string filename = ruta + std::string("/file.config");
    std::ifstream myfile(filename.c_str());

    if (myfile.is_open()) {
 std::string linea_archivo;
 std::string concatenado;

         while ( std::getline (myfile,linea_archivo) ) {
     std::size_t si_ejecuta = linea_archivo.find("1",0,1); //Buscamos si tiene un 1. Ejecuta.
     std::size_t no_ejecuta = linea_archivo.find("2",0,1); //Buscamos en la línea actual el carácter 2. No ejecuta.
             std::size_t argumento = linea_archivo.find("#",0,1); //Buscamos en la línea actual el '#'. Es comentario.
                 if (no_ejecuta == 0) {
          ejecutado = true; //Si encuentra un 2, no queremos que ejecute. Marcamos flag.
      printf("%s\tIgnorando test según se especifica en el fichero de configuración.\n",posstr);
                      break; //No leer más líneas
     } else if (si_ejecuta != 0) { //Si la línea es 1, ejecuta como sin argumentos y no entra en el bucle
          if (argumento != 0){ //Con argumentos
                               //Descarta línea con la '#'. La línea restante la usa como argumento y concatena.
           con_argumento = true;
           concatenado = comandocompleto_argumentos + ' ' + linea_archivo + " > /dev/null";
                               break; //No leer más líneas
          }
     }
 }//while
 //printf("Concatenado: %s.\n",concatenado.c_str());
         myfile.close();
 if (con_argumento == true) {
      ejecutado = true;
              BOOST_TEST(system(concatenado.c_str())==0);
      //printf("con args.\n");
      free(posstr);
         }
    }

    if (ejecutado == false){ //Si no hay fichero de configuración, se ejecuta sin argumentos.
    BOOST_TEST(system(comandocompleto.c_str())==0);
```

Imagen 21. Búsqueda de fichero de configuración y ejecución según su contenido.

- El programa, comprueba que el valor de retorno del programa, y lo analiza gracias a la librería BOOST. En el caso del programa diseñado en este TFM, se comprueba que los tests devuelvan valor 0 si se han ejecutado correctamente. En caso contrario, se genera un error, el cual se muestra por pantalla, tal y como se muestra en la imagen inferior:

Imagen 22. Comprobación de test con error.

- Además, el programa solo va a ejecutar archivos que cumplan con el requisito de ser ejecutables. Para ello, se ha creado la siguiente

función, la cual comprueba si la ruta completa absoluta que se le pasa por parámetro cumple los requisitos de ser ejecutable:

```
bool isExecutable(char *fname, struct dirent *ent)
{
  struct stat sb;

  if (stat(fname, &sb) == 0 && sb.st_mode & S_IXUSR) return true;
  else return false;
}
```

Imagen 23. Función isExecutable del programa.

En la siguiente imagen, se muestra un organigrama con el flujo del programa y su funcionamiento según la casuística abordada:

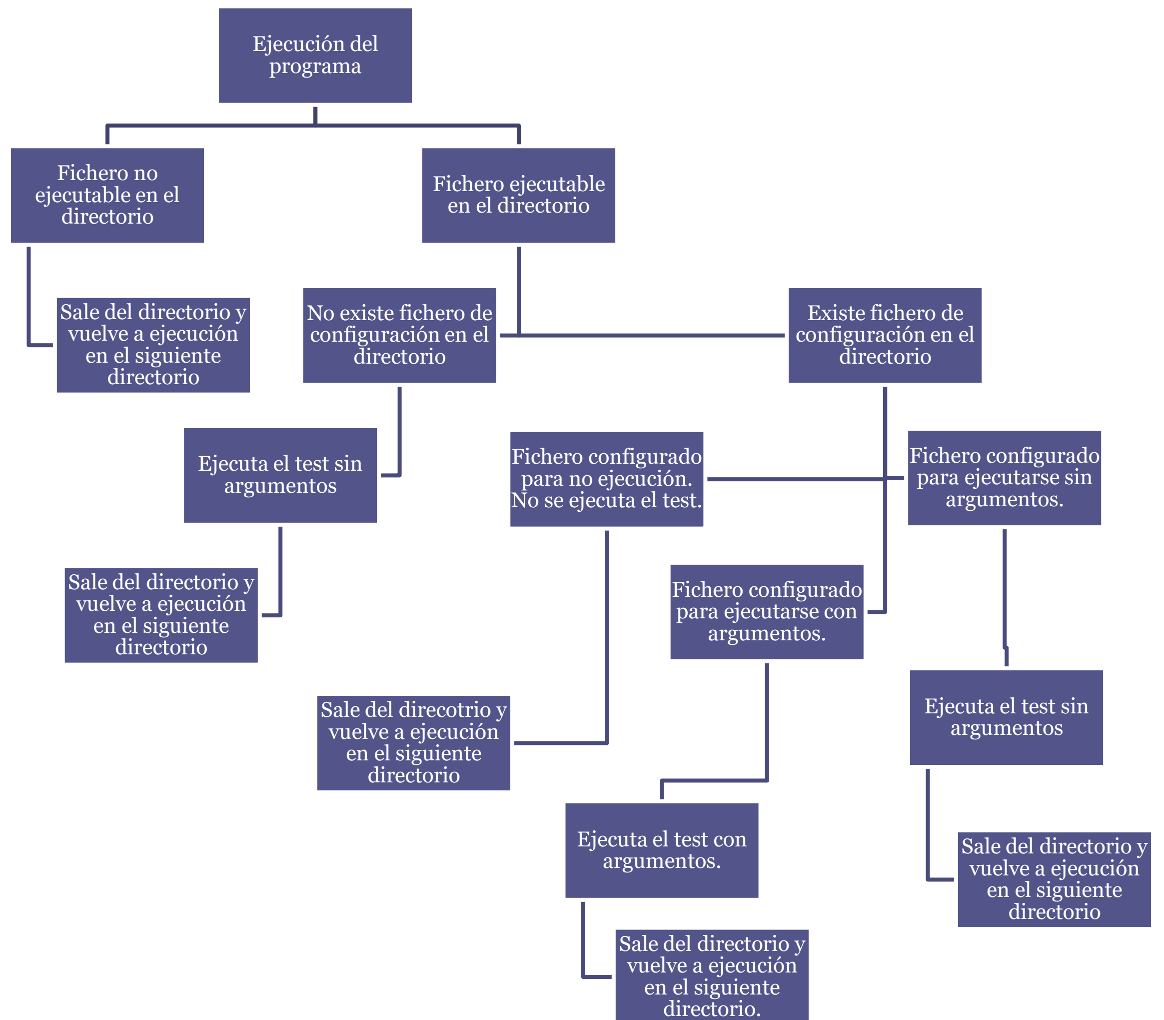

Imagen 24. Organigrama de la ejecución del programa.

En el capítulo de Anexo se encuentra todo el programa escrito y expuesto en su totalidad. En él se explica con todo lujo de detalles con los comentarios, cual es el funcionamiento del programa en su conjunto.

Ejemplos de estructura de ficheros de configuración

Puede haber tres opciones diferentes de comportamiento dentro de los ficheros de configuración, y son los siguientes:

```
*Ejemplo de ejecución con argumentos*
1
# Argumento para clock
--device=0

*Ejemplo de no ejecución*
2

*Ejemplo de ejecución sin argumentos*
1
```

5.3 Ejecución del programa

La ejecución del programa diseñado y programado en este TFM sigue el siguiente patrón:

- Nombre del programa ejecutable, típicamente "tests_unitarios.cpp".
- Parámetros de BOOST. Puede pasarse en blanco o bien pasarle un parámetro de esta librería. Destacan los siguientes:
    - Auto_start_dbg: Inicia automáticamente el depurador ("debugger") en caso de recibir una señal de fallo a nivel de sistema.
    - Build_info: Muestra por pantalla información de compilación de la librería.
    - Catch_system_errors: Permite cambiar entre las opciones de cachear o ignorar los errores de sistema.
    - Detect_memory_leaks: Activa / desactiva la detección de pérdidas de memoria.
    - Random. Este parámetro es utilizado en la ejecución del programa, aunque no es realmente necesario. Permite cambiar entre orden secuencial y orden aleatorio en la ejecución de los tests unitarios. En nuestro programa, al ejecutar muchos tests de una sola vez, el programa indicaba un "warning". Al usar este parámetro en la ejecución, el programa se ejecuta sin el "warning". De todas formas, no es estrictamente necesario usarlo ya que el resultado es el mismo, pero siempre queda más limpio un código que se ejecuta sin ninguna alerta.

- Ruta donde comenzar a buscar ficheros ejecutables recursivamente, y bajar a niveles inferiores del árbol de directorios siguiendo este mismo patrón.

6. Experimentos

Se han realizado una serie de experimentos con el software programado, testando, por una parte, samples de CUDA en mi máquina personal, usando el modo loopback del middleware rCUDA, para simular un clúster y así recibir respuesta del demonio de rCUDA; y por otra parte, testando software independiente de los samples CUDA y NVIDA en un clúster de la Universitat Politècnica de València.

6.1 Experimentos con samples de CUDA

En este experimento se va a testear el middleware rCUDA usando los samples de CUDA como tests. Primeramente, desde el directorio donde está alojado el programa, se ha ejecutado el programa, el cual lanza unos tests desde la ruta indicada, descendiendo en directorios inferiores, con la siguiente orden:

```
./tests_unitarios --random -- /usr/local/cuda-
6.0/samples_2/
```

El árbol de directorios por el que el programa ha ido buscando y ejecutando binarios y buscando, leyendo y tratando ficheros de configuración es el siguiente:

```
/usr/local/cuda-6.0/samples_2
├── 0_Simple
│   ├── clock
│   │   ├── clock.cu
│   │   ├── CMakeLists.txt
│   │   ├── file.config
│   │   ├── Makefile
│   │   ├── NsightEclipse.xml
│   │   └── readme.txt
│   └── vectorAdd
│       ├── CMakeLists.txt
│       ├── Makefile
│       ├── NsightEclipse.xml
│       ├── readme.txt
│       ├── vectorAdd
│       ├── vectorAdd.cu
│       └── vectorAdd.o
├── 1_Utilities
│   ├── bandwidthTest
│   │   ├── bandwidthTest
│   │   ├── bandwidthTest.cu
│   │   ├── bandwidthTest.o
│   │   ├── CMakeLists.txt
│   │   ├── file.config
│   │   ├── Makefile
│   │   ├── NsightEclipse.xml
│   │   └── readme.txt
│   ├── deviceQuery
│   │   ├── CMakeLists.txt
│   │   ├── deviceQuery
│   │   ├── deviceQuery.cpp
│   │   ├── deviceQuery.o
│   │   ├── file.config
│   │   ├── Makefile
│   │   ├── NsightEclipse.xml
│   │   └── readme.txt
│   ├── deviceQueryDrv
│   │   ├── CMakeLists.txt
│   │   ├── deviceQueryDrv
│   │   ├── deviceQueryDrv.cpp
│   │   ├── deviceQueryDrv.o
│   │   ├── file.config
│   │   ├── Makefile
│   │   ├── NsightEclipse.xml
│   │   └── readme.txt
│   └── p2pBandwidthLatencyTest
│       ├── Makefile
│       ├── NsightEclipse.xml
│       ├── p2pBandwidthLatencyTest
│       ├── p2pBandwidthLatencyTest.cu
│       ├── p2pBandwidthLatencyTest.o
│       └── readme.txt
└── arbol.txt
```

Los ficheros de configuración usados para cada uno de los tests, y alojados en el mismo directorio que en el que se ha ejecutado cada uno de los tests, son los siguientes:

clock

```
1
# Argumento para clock
--device=0
```

vectorAdd

```
No tiene fichero de configuración. Se ejecuta sin argumentos.
```

bandwidthTest

```
1
# Prueba
--mode=incorrect
```

Nota: esta ejecución está forzada para que de error, y el test nos lo indique.

deviceQuery

```
1
```

deviceQueryDrv

```
2
```

p2pBandwidthLatencyTest

```
No tiene fichero de configuración. Se ejecuta sin argumentos.
```

El programa ha generado una salida por pantalla, en la cual se muestra la ejecución de todos los tests y su resultado:

```
root@isma-desktop:/usr/local/cuda-6.0/samples# ./tests_unitarios --random -- /usr/local/cuda-6.0/samples_2/

Ejecutando tests en la ruta: /usr/local/cuda-6.0/samples_2/

Running 1 test case...
      Ejecutando test: /usr/local/cuda-6.0/samples_2/1_Utilities/bandwidthTest/bandwidthTest
tests_unitarios.cpp(190): error: in "cuentaArchivos": check system(concatenado.c_str())==0 has failed [256 != 0]

      Ejecutando test: /usr/local/cuda-6.0/samples_2/1_Utilities/deviceQueryDrv/deviceQueryDrv
        Ignorando test según se especifica en el fichero de configuración.

      Ejecutando test: /usr/local/cuda-6.0/samples_2/1_Utilities/p2pBandwidthLatencyTest/p2pBandwidthLatencyTest

      Ejecutando test: /usr/local/cuda-6.0/samples_2/1_Utilities/deviceQuery/deviceQuery


      Ejecutando test: /usr/local/cuda-6.0/samples_2/0_Simple/vectorAdd/vectorAdd



*** 1 failure is detected in the test module "Master Test Suite"
```

Imagen 25. Ejecución del programa con samples de CUDA.

6.2 Experimentos en clúster de la UPV

Como se ha explicado anteriormente, el software diseñado y programado en este TFM es compatible, además de con los samples de CUDA y el testeo con rCUDA, con tests de otros tipos de software. Para demostrarlo, se han realizado una serie de tests de software independiente de los samples de CUDA. Estas pruebas se han realizado en un clúster gracias al apoyo del grupo GAP (Parallel Architectures Group) de la Universitat Politècnica de València, que decorosamente ha prestado sus instalaciones para esta parte de los experimentos del TFM.

Para ejecutar el test, se realiza según los comandos de la parte inferior. Usamos el parámetro "random" de BOOST, tal y como se ha explicado unos párrafos más atrás:

```
-bash-4.1$ ./tests_unitarios --random --
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75
```

Desde la ruta indicada por parámetro en la ejecución, se ha descendido por los directorios inferiores. El árbol de directorios por el que el programa ha ido buscando y ejecutando binarios y buscando, leyendo y tratando ficheros de configuración es el siguiente:

```
├── BarraCUDA
│   ├── bin
│   │   ├── barracuda
│   │   ├── test.config
│   │   └── manyfish
│   └── TESTS ->
/nfs/opt/APPS/APPS/BarraCUDA/Barracuda_0.6.2_beta/barracuda+CUDA65/TESTS/
├── CUDA-MEME
│   ├── bin
│   │   ├── cuda-meme
│   │   └── test.config
│   └── TESTS -> /nfs/opt/APPS/APPS/CUDA-MEME/3.0.15/CUDA70/TESTS
├── CUDASW++
│   ├── bin
│   │   ├── cudasw
│   │   └── test.config
│   └── TESTS -> /nfs/opt/APPS/APPS/CUDASW++/3.1.1/CUDA75/TESTS
├── GPU-BLAST
│   ├── bin
│   │   ├── blastp
│   │   └── test.config
│   └── TESTS -> /nfs/opt/APPS/APPS/GPU-BLAST/1.1_ncbi-blast-2.2.28-CUDA75/TESTS
├── GPU-LIBSVM
│   ├── bin
│   │   ├── test.config
│   │   ├── svm-train-gpu
│   │   └── train_set.model
│   └── TESTS -> /nfs/opt/APPS/APPS/GPU-LIBSVM/3.18/TESTS
├── LAMMPS
│   ├── bin
│   │   ├── test.config
│   │   └── lmp_mpi
│   └── TESTS -> /nfs/opt/APPS/APPS/LAMMPS/lammps-14May16/mvapich2+CUDA75/TESTS
├── MAGMA
│   └── bin
│       ├── test.config
│       └── testing_dlarfb_gpu
└── NAMD
    ├── bin
    │   ├── test.config
    │   └── namd2
    └── TESTS -> /nfs/opt/APPS/APPS/NAMD/2.11/multicore+GPU+CUDART7.5/TESTS
```

Los ficheros de configuración usados para cada uno de los tests, y alojados en el mismo directorio que el ejecutable, tal y como se observa en el árbol de directorios anterior, son los siguientes:

BarraCuda

```
# Fichero configuración BarraCUDA
1
aln
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/BarraCUDA/TESTS/database/human_g1k_v37
.fasta
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/BarraCUDA/TESTS/queries/37mer_alt_1.fa
stq > /dev/null
```

CUDA-MEME

```
# Fichero configuración CUDA-MEME
1
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/CUDA-
MEME/TESTS/nrsf_testcases/nrsf_500.fasta -dna -mod oops -maxsize 500000
```

CUDASW

```
# Fichero configuración CUDASW++
1
-query
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/CUDASW++/TESTS/Queries/Q9UKN1.fasta  -
db
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/CUDASW++/TESTS/DataBases/uniprot_sprot
.fasta -use_single 0
```

GPU-BLAST

```
# Fichero configuración GPU-BLAST
1
-db database/sorted_env_nr -query /nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/GPU-
BLAST/TESTS/queries/SequenceLength_00004998.txt -num_threads 1 -gpu t
```

GPU-LIBSVM

```
# Fichero configuración GPU-LIBSVM
1
-h 0 -v 10 /nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/GPU-
LIBSVM/TESTS/svmguide1.t
```

LAMMPS

```
# Fichero configuración LAMMPS
1
-c on -sf cuda -in
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/LAMMPS/TESTS/bench/in.eam -var x 5 -
var y 5 -var z 5
```

MAGMA

```
# Fichero configuración MAGMA
1
--range 1088:4160:1024
```

NAMD

```
# Fichero configuración NAMD
1
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/NAMD/TESTS/apoa1/apoa1.namd
```

Al igual que en el apartado anterior, el programa genera una salida por pantalla, en la cual se muestra la ejecución de todos los tests y el resultado de cada uno de ellos. Si no muestra nada tras la ejecución del test, significa que el programa ha recibido como valor de retorno un 0, lo que indica que el test ha sido superado. En caso contrario, mostraría un error por pantalla y sumaría 1 al contador de errores, el cual muestra al final de la ejecución del programa:

```
Ejecutando tests en la ruta:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75

Running 1 test case...


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/NAMD/bin/namd2


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/GPU-LIBSVM/bin/svm-
train-gpu


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/LAMMPS/bin/lmp_mpi


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/GPU-BLAST/bin/blastp


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/MAGMA/bin/testing_dla
rfb_gpu


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/CUDA-MEME/bin/cuda-
meme


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/BarraCUDA/bin/barracu
da


      Ejecutando test:
/nfs/opt/APPS/APPS/rCUDA/UNIT_TESTS/CUDA75/CUDASW++/bin/cudasw
```

```
*** No errors detected
```

Esta salida por pantalla denota que el software programado para este TFM se ha ejecutado correctamente, y que al ejecutarse a su vez se han ejecutado los tests indicados, siguiendo los ficheros de configuración de sus rutas, si los hubiere, no hallándose ningún error.

6.3 Experimento con el middleware rCUDA

En esta última parte se va a probar el middleware rCUDA. Para ello, hay que ejecutar rCUDA de manera que espere los resultados de los tests ejecutados con los samples de CUDA.

Para probar que el acceso con rCUDA funciona correctamente, se ha lanzado otro test, previa configuración de variables de entorno.
Este test se ha compilado con el siguiente flag especial, para que pueda utilizarse de una manera compartida. rCUDA se ejecuta en modo loopback, esto es, para que simule el acceso a una GPU virtual cuando realmente está utilizando nuestra IP local 127.0.0.1:

```
make EXTRA_NVCCFLAGS=--cudart=shared
```

La salida del programa se muestra en la imagen siguiente. El test se ha ejecutado de una manera correcta, mostrando por la pantalla el resultado. De él se denota que no se han producido errores:

```
root@isma-desktop:/usr/local/cuda-6.0/samples# export LD_LIBRARY_PATH=/usr/local/rCUDAv15.07-CUDA6.0/lib/:$LD_LIBRARY_PATH
root@isma-desktop:/usr/local/cuda-6.0/samples# export RCUDA_DEVICE_0="127.0.0.1"
root@isma-desktop:/usr/local/cuda-6.0/samples# export RCUDA_DEVICE_COUNT="0"
root@isma-desktop:/usr/local/cuda-6.0/samples# ./tests_unitarios --random -- /usr/local/cuda-6.0/samples/1_Utilities/deviceQuery

Ejecutando tests en la ruta: /usr/local/cuda-6.0/samples/1_Utilities/deviceQuery

Running 1 test case...
  Ejecutando test: /usr/local/cuda-6.0/samples/1_Utilities/deviceQuery/deviceQuery

*** No errors detected
```

Imagen 26: Ejecución del programa con un sample de CUDA compilado para que sea compartido.

En la siguiente imagen, se ha ejecutado el demonio de rCUDA, con el siguiente commando:

```
./rCUDAd -ilv
```

El parámetro “l” indica que se ejecuta en modo loopback, y el parámetro “v” indica que se ejecuta en modo “verbose”, con lo cual muestra por pantalla todo lo que envía y recibe.

Al ejecutar los tests de la imagen anterior, el demonio ha respondido por la salida estándar los siguientes mensajes. De ellos se denota que el servidor ha recibido la petición correctamente, se ha inicializado CUDA en el dispositivo 0 (la GPU del equipo), y se han realidado las operaciones oportunas, recibiendo de CUDA los mensajes correctamente. Finalmente, al acabar la ejecución del test anterior, rCUDA nos indica que la aplicación remota ha finalizado, es decir, el programa que realiza los tests ha acabado correctamente, y se queda a la espera de recibir más peticiones.

```
root@isma-desktop:/usr/local/rCUDAv15.07-CUDA6.0/bin# clear
root@isma-desktop:/usr/local/rCUDAv15.07-CUDA6.0/bin# ./rCUDAd -ilv
rCUDAd v15.07
Copyright 2009-2015 UNIVERSITAT POLITECNICA DE VALENCIA. All rights reserved.
rCUDAd[2795]: Using rCUDAcommTCP.so communications library.
rCUDAd[2795]: Server daemon succesfully started.
rCUDAd[2798]: Trying device 0...
rCUDAd[2798]: CUDA initialized on device 0.
rCUDAd[2798]: Connection established with localhost.
rCUDAd[2798]: 'cudaDeviceReset' received.
rCUDAd[2806]: Trying device 0...
rCUDAd[2806]: CUDA initialized on device 0.
rCUDAd[2798]: Remote application finished.
```

Imagen 27: Salida del demonio de rCUDA con mensajes satisfactorios.

7. Conclusiones

Tal y como se vio en la asignatura CPD del máster (Centros de Procesamiento de Datos y Virtualización), hoy en día la virtualización es una realidad más que asentada en nuestro entorno. Una realidad, y cada vez más una necesidad: gracias a ella podemos replicar y tener equipos muy potentes sin necesidad de invertir millones de euros en comprar hardware. La virtualización de GPUs, sobre todo, está marcando un antes y un después en la capacidad de cómputo, consiguiendo que con muy pocos recursos hardware se consigan resultados muy satisfactorios.

Gracias al uso de la herramienta middleware rCUDA, podemos virtualizar el acceso a GPUs. Pero rCUDA necesita ser verificado: por ello, en este TFM se ha realizado un programa que verifica rCUDA automáticamente. Además, este programa permite realizar tests a nodos virtuales desde un mismo equipo, con lo cual se ahorra en costes de mantenimiento de servidores, de hardware, de software, etc.

Gracias a la automatización de los tests de software, se produce un gran ahorro de tiempo y recursos (a nivel de hardware y, especialmente, ahorro de recursos humanos), lo que se traduce en un gran ahorro económico, cosa que hoy en día resulta primordial.

El fin de este software es facilitar el testing del middleware rCUDA, el permite, “con un solo clic”, acceder a distintas GPUs sitas en nodos diferentes aunque también es extrapolable a testear software de otros ámbitos, tal y como se demuestra en capítulos anteriores testando software de terceros, debido a que se ha desarrollado con una amplia visión de futuros usos.

En un futuro, este software será útil para programas que necesiten tests analizando la salida estándar que generan. Asimismo, se podría modificar y mejorar para que analizara diferentes salidas de programa según el tipo de tests que generara, ya que no siempre todos los tests devuelven el mismo valor. Esto se podría programar en la lectura del fichero de configuración, para que el programa leyera de él lo que sería una salida o respuesta satisfactoria, y la comparara usando la librería BOOST, con la salida que ha generado el programa al ejecutar el test. Así las posibilidades de validar software serían aún mucho más amplias de las que ya son.

8. Bibliografía y referencias

Bibliografía

[1]. Open-source electronic prototyping platform enabling users to create interactive electronic objects. [En línea] Obtenido de: https://www.arduino.cc.

[2]. Computadora Industrial Abierta Argentina. [En línea] Obtenido de: http://www.proyecto-ciaa.com.ar/devwiki/lib/exe/fetch.php?media=desarrollo:firmware:breve_introduccio_n_a_osek-vdx.pdf.

[3]. The high-quality transcription factor binding profile database. [En línea] Obtenido de https://www.jaspar.jp/english/index_e.php.

[4]. Open Source High Performance Computing. [En línea] Obtenido de https://www.open-mpi.org/.

[5]. VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds. [En línea] Obtenido de http://www.vmware.com/es.html.

[6]. The Xen Project, the powerful open source industry standard for virtualization. [En línea] Obtenido de https://www.xenproject.org/.

[7]. Oracle VM VirtualBox. [En línea] Obtenido de https://www.virtualbox.org/.

[8]. AWS – Cloud Computing. [En línea] Obtenido de https://aws.amazon.com/es/.

[9]. Google Cloud Platform. [En línea] Obtenido de https://cloud.google.com.

[10]. Microsoft Azure: plataforma y servicios de informática en la nube. [En línea] Obtenido de https://azure.microsoft.com/es-es/.

[11]. Citrix: Asegure, movilice y optimice la entrega de aplicaciones y datos con Citrix [En línea] Obtenido de https://citrix.es.

[12]. XenServer – Open Source Server Virtualization. [En línea] Obtenido de http://xenserver.org/.

[13]. NVIDIA Corporation – Titan X. [En línea] Obtenido de http://www.nvidia.es/object/geforce-gtx-titan-x-es.html.

[14]. NVIDIA Corporation – Tesla. [En línea] Obtenido de http://www.nvidia.es/object/tesla-supercomputer-workstations-es.html.

[15]. NVIDIA Corporation – Tesla. [En línea] Obtenido de http://www.nvidia.com/object/tesla-p100.html.

[16]. NVIDIA Corporation – Grid Technology. [En línea] Obtenido de http://www.nvidia.com/object/grid-technology.html.

[17]. How people build software: GitHub. [En línea] Obtenido de https://github.com/zillians/platform_manifest_vgpu.

[18]. Remot CUDA software developed by www.upv.es. [En línea] Obtenido de htttp://rcuda.net.

[19] Liang, T., & Chang, Y. (2011). *GridCuda: A Grid Enabled CUDA Programming Toolkit.*

[20] Oikawa, M., Kawai, A., Nomura, K., Yasuoka, K., Yoshikawa, K., & Narumi, T. (2012). *DS-CUDA: A Middleware to Use Many GPUs in the Cloud Enviroment, Networkinkg Storage and Analysis.*

[21] Iserte, S., Prades, J., Reaño, C., & Silla, F. (2016). *Increasing the Performance of Data Centers by Combining Remote GPU Virtualization with Slurm.*

[22] Reaño, C., & Silla, F. (2015). *ON THE DEPLOYMENT AND CHARACTERIZATION OF CUDA TEACHING LABORATORIES.*

[23] Pérez, F., Reaño, C., & Silla, F. (2016). *Providing CUDA Acceleration to KVM Virtual Machines in InfiniBand Clusters with rCUDA.*

[24] Prades, J., Reaño, C., & Silla, F. (2016). *CUDA Acceleration for Xen Virtual Machines in InfiniBand Clusters with rCUDA.*

[25]. (s.f.). Obtenido de http://developer.download.nvidia.com/compute/cuda/7.5/Prod/docs/sidebar/CUDA_Installation_Guide_Linux.pdf.

[26] Pressman. (2005). *Capítulo 17: Técnicas de Prueba del Software. en: Ingeniería del Software.* Obtenido de Pressman.

[27]. A little madness – C Unit testing with Boot.test. [En línea] Obtenido de http://www.alittlemadness.com/2009/03/31/c-unit-testing-with-boosttest/.

9. Anexos

El programa ha sido nombrado “tests_unitarios.cpp”. El código fuente programado es el siguiente:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>
#include <errno.h>
#include <boost/test/included/unit_test.hpp>
#include <cstddef>
#include <algorithm>

using namespace boost::unit_test;

/* Función para devolver un error en caso de que ocurra */
void error(const char *s);

/* Calculamos el tamaño del archivo */
long fileSize(char *fname);

/* Sacamos el tipo de archivo haciendo un stat(), es como el stat de la línea de
comandos */
unsigned char statFileType(char *fname);

/* Intenta sacar el tipo de archivo del ent */
unsigned char getFileType(char *ruta, struct dirent *ent);

/* Devuelve true si el archivo es un ejecutable */
bool isExecutable(char *ruta, struct dirent *ent);

/* Obtiene el nombre del fichero con la ruta completa */
char *getFullName(char *ruta, struct dirent *ent);

/* Obtiene el comando de ejecución del test para casos con argumentos */
std::string getFullCommand(char *ruta, struct dirent *ent);

/* Obtiene el comando de ejecución del test para casos con argumentos */
std::string getFullCommand_ArgumentsCase(char *ruta, struct dirent *ent);

/* Genera una cadena de espacios, para dibujar el árbol */
char *generaPosStr(int niv);

/* Función principal, que ejecuta los tests recursivamentes */
char *ruta_main;
int niv_main;
void cuentaArchivos();
unsigned cuentaArchivos2(char *ruta, int niv);

test_suite* init_unit_test_suite( int argc, char* argv[] )
{
  unsigned num;

  if (argc != 2)
    {
```

```
      error("Uso: ./directorio_2 <ruta>\n");
    }
  printf("\nEjecutando tests en la ruta: %s\n\n", argv[1]);
  ruta_main=argv[1];
  niv_main=1;
  framework::master_test_suite().add( BOOST_TEST_CASE( &cuentaArchivos ) );

  return EXIT_SUCCESS;
}

void error(const char *s)
{
  /* perror() devuelve la cadena S y el error (en cadena de caracteres) que tenga
  errno */
  perror (s);
  exit(EXIT_FAILURE);
}

//Nos trae el nombre completo del fichero
char *getFullName(char *ruta, struct dirent *ent)
{
  char *nombrecompleto;
  int tmp;

  tmp=strlen(ruta);
  nombrecompleto=(char *)malloc(tmp+strlen(ent->d_name)+2); /* Sumamos 2, por el
\0 y la barra de directorios (/) no sabemos si falta */
  if (ruta[tmp-1]=='/')
    sprintf(nombrecompleto,"%s%s", ruta, ent->d_name);
  else
    sprintf(nombrecompleto,"%s/%s", ruta, ent->d_name);

  return nombrecompleto;
}

//Función que devuelve la ruta completa para ejecutar el test, para casos con
argumentos en el fichero de configuración.
std::string getFullCommand_ArgumentsCase(char *ruta, struct dirent *ent)
{
  std::string str1(ruta);
  std::string str2(ent->d_name);
  std::string nombrecompleto_ArgumentsCase = str1 + "/" + str2;

  return nombrecompleto_ArgumentsCase;
}

//Función que devuelve la ruta completa para ejecutar el test, para casos sin
argumentos en el fichero de configuración, o sin
//fichero de configuración en la ruta.
std::string getFullCommand(char *ruta, struct dirent *ent)
{
  std::string str1(ruta);
  std::string str2(ent->d_name);
  std::string nombrecompleto = str1 + "/" + str2 + " > /dev/null";

  return nombrecompleto;
}

//Función para generar los espacios y dotar de estructura de árbol a la ejecución
del programa.
char *generaPosStr(int niv)
{
```

```
  int i;
  char *tmp=(char *)malloc(niv*2+1);    /* Dos espacios por nivel más terminador
*/
  for (i=0; i<niv*2; ++i)
    tmp[i]=' ';
  tmp[niv*2]='\0';
  return tmp;
}

//Función utilizada desde el método main, la cual llama recursivamente a la
función cuentaArchivos2.
void cuentaArchivos()
{
   cuentaArchivos2(ruta_main, niv_main);
}

//Función madre, la cual realiza el grueso del programa: tests, lectura de
ficheros, etc.
unsigned cuentaArchivos2(char *ruta, int niv)
{
  /* Con un puntero a DIR abriremos el directorio */
  DIR *dir;
  /* en *ent habrá información sobre el archivo que se está "sacando" a cada
momento */
  struct dirent *ent;
  unsigned numfiles=0;          /* Ficheros en el directorio actual */
  unsigned char tipo;       /* Tipo: fichero /directorio/enlace/etc */
  char *nombrecompleto;     /* Nombre completo del fichero */
  std::string comandocompleto;     /* Comando completo del test para casos sin
argumentos */
  std::string comandocompleto_argumentos; /* Comando completo del test para casos
con argumentos */
  char *posstr;         /* Cadena usada para posicionarnos horizontalmente */
  dir = opendir (ruta);

  /* Miramos que no haya error */
  if (dir == NULL)
    error("No puedo abrir el directorio");

  while ((ent = readdir (dir)) != NULL)
    {
      if ( (strcmp(ent->d_name, ".")!=0) && (strcmp(ent->d_name, "..")!=0) )
    {
      nombrecompleto=getFullName(ruta, ent);
      tipo=getFileType(nombrecompleto, ent);
      if (tipo==DT_REG)
        {
          ++numfiles;
          if(isExecutable(nombrecompleto, ent))
          {
             posstr=generaPosStr(niv);
             printf("%sEjecutando test: %s\n", posstr, nombrecompleto);
             comandocompleto=getFullCommand(ruta, ent); //Sin argumetnos
             comandocompleto_argumentos=getFullCommand_ArgumentsCase(ruta, ent);
             //Con argumentos

             bool ejecutado = false; //Para comprobar si ha entrado en a=1, no
             haga una segunda ejecución del test
             bool con_argumento = false;
             bool sin_argumento = false;
             std::string filename = ruta + std::string("/test.config");
             std::ifstream myfile(filename.c_str());
```

```
            if (myfile.is_open()) {
                   std::string linea_archivo;
                   std::string concatenado;

                   while ( std::getline (myfile,linea_archivo) ) {
                          std::size_t si_ejecuta = linea_archivo.find("1",0,1);
                          //Buscamos si tiene un 1. Ejecuta.
                          std::size_t no_ejecuta = linea_archivo.find("2",0,1);
                          //Buscamos en la línea actual el carácter 2. No
                          ejecuta.
                          std::size_t argumento = linea_archivo.find("#",0,1);
                          //Buscamos en la línea actual el '#'. Es comentario.
                          if (no_ejecuta == 0) {
                                 ejecutado = true;
                          //Si encuentra un 2, no queremos que ejecute. Marcamos
                          flag.
                          printf("%s\tIgnorando test según se especifica en el
                          fichero de configuración.\n",posstr);
                           break; //No leer más líneas
                          } else if (si_ejecuta != 0) {
                          //Si la línea es 1, ejecuta como sin argumentos y no
                          entra en el bucle
                          if (argumento != 0){ //Con argumentos
                          //Descarta línea con la '#'. La línea restante la usa
                          como argumento y concatena.
                                 con_argumento = true;
                                 concatenado = comandocompleto_argumentos + '
             '+ linea_archivo + " > /dev/null";
                                 break; //No leer más líneas
                          }
                        }
                      }//while
                //printf("Concatenado: %s.\n",concatenado.c_str());
                myfile.close();
                if (con_argumento == true) {
                    ejecutado = true;
                    BOOST_TEST(system(concatenado.c_str())==0);
                    //printf("con args.\n");
                    free(posstr);
                }
            }

            if (ejecutado == false){
             //Si no hay fichero de configuración, se ejecuta sin argumentos.
             BOOST_TEST(system(comandocompleto.c_str())==0);
             //printf("sin args o con valor 1 en el fichero.\n");
             free(posstr);
            }
          }
        }
      else if (tipo==DT_DIR)
      {
         posstr=generaPosStr(niv);
         cuentaArchivos2(nombrecompleto, niv+1);
         printf("\n");
         free(posstr);
      }
      free(nombrecompleto);
    }
  }
closedir (dir);
```

```
}

//Función que recupera y devuelve el tipo de archivo.
unsigned char getFileType(char *nombre, struct dirent *ent)
{
  unsigned char tipo;

  tipo=ent->d_type;
  if (tipo==DT_UNKNOWN)
    {
      tipo=statFileType(nombre);
    }

  return tipo;
}

//Función que indica si un archivo es ejecutable o no.
bool isExecutable(char *fname, struct dirent *ent)
{
  struct stat sb;

  if (stat(fname, &sb) == 0 && sb.st_mode & S_IXUSR) return true;
  else return false;
}

//Función que busca el tipo de archivo.
unsigned char statFileType(char *fname)
{
  struct stat sdata;

  /* Intentamos el stat() si no funciona, devolvemos tipo desconocido */
  if (stat(fname, &sdata)==-1)
    {
      return DT_UNKNOWN;
    }

  switch (sdata.st_mode & S_IFMT)
    {
    case S_IFBLK:  return DT_BLK;
    case S_IFCHR:  return DT_CHR;
    case S_IFDIR:  return DT_DIR;
    case S_IFIFO:  return DT_FIFO;
    case S_IFLNK:  return DT_LNK;
    case S_IFREG:  return DT_REG;
    case S_IFSOCK: return DT_SOCK;
    default:       return DT_UNKNOWN;
    }
}
```