Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

Reduciendo el coste económico de las prácticas de CUDA manteniendo la calidad del aprendizaje

Carlos Reaño y Federico Silla
Departament d'Informàtica de Sistemes i Computadors (DISCA)
Escola Tècnica Superior d'Enginyeria Informàtica (ETSINF)
Universitat Politècnica de València (UPV)
carregon@gap.upv.es, fsilla@disca.upv.es

Resumen

La computación de propósito general con tarjetas gráficas se basa en el uso de estas tarjetas (GPUs) para realizar cálculos computacionales que tradicionalmente son realizados por los procesadores (CPUs). Debido al creciente uso de las GPUs, es importante que los planes de estudio de informática incluyan los fundamentos de la computación paralela con GPUs, al tiempo que se equipan los laboratorios docentes con GPUs a un coste razonable. En este sentido, instalar GPUs en todos los ordenadores del laboratorio puede resultar costoso a nivel económico, mientras que compartir un servidor remoto con GPU entre los estudiantes puede derivar en unas malas condiciones de aprendizaje.
En este trabajo proponemos una solución eficaz a este problema: el uso de la tecnología rCUDA (CUDA remoto), que permite a las aplicaciones de un ordenador utilizar, de forma concurrente y transparente, GPUs instaladas en servidores remotos. De esta manera los estudiantes pueden, desde sus puestos de trabajo, compartir una misma GPU instalada en un servidor remoto sin tener que iniciar sesión en el mismo. Para demostrar que nuestra propuesta es factible, presentamos experimentos en un escenario real que muestran cómo el coste del laboratorio es notablemente reducido, mientras que la calidad del aprendizaje se mantiene.

Abstract

General-Purpose computing on Graphics Processing Units consists in using Graphics Processing Units (GPUs) to perform the computation of applications traditionally handled by regular processors (CPUs). Due to their increasing use, it is important that Computer Engineering and Computer Science curricula include the basics of this new computing trend. As regards the practical part of the training, one major issue is how to introduce GPUs into a laboratory: buying GPUs for all the workstations of the lab may be too expensive, whereas installing one GPU in a server and requesting the students to log into this server may lead to a low teaching quality due to its associated overhead.
In this paper we suggest a new solution to introduce GPUs into a laboratory: the rCUDA (remote CUDA) framework, which allows applications running in a computer to use GPUs installed in remote servers. Hence, students will be capable of sharing a remote GPU (concurrently and transparently) from their local workstations in the lab, without logging into the server. To prove that our approach is possible, we show experiments in a real laboratory. The experiments demonstrate that our proposal reduces the cost of the laboratory, whereas the teaching quality still remains.

Palabras clave

CUDA, aprendizaje, laboratorio, costes.

1. Introducción

La computación paralela ha sido tradicionalmente incluida en los planes de estudio de informática para enseñar a los estudiantes cómo hacer frente a retos impuestos por problemas complejos. Estos problemas requieren de una gran cantidad de recursos computacionales, los cuales deben colaborar entre ellos para conseguir una computación de altas prestaciones.

Durante los últimos años, las unidades de procesamiento gráfico (*Graphics Processing Units*—GPUs) han sido ampliamente utilizadas para acelerar aplicaciones en áreas tan diversas como, por ejemplo, análisis de datos [1], física y química [2], análisis de imágenes [3], finanzas [4], algebra [5], dinámica de fluidos [6], etc. Para este tipo de aplicaciones, el uso de GPUs se está generalizando debido a la buena relación coste/prestaciones que ofrecen. Además, el uso de GPUs también resulta ventajoso desde un punto de vista energético, dado que ofrecen una excelente relación Gflops/watt. Por este motivo los primeros ordenadores

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

de la lista Green500[1] inclu
no obstante, que hay ciertos
el uso de GPUs no es la mej

Debido al notable increm
do el uso de GPUs, es imp
estudio de informática inclu
computación paralela con G
que OpenCL [7] es el estánc
zado para programar GPUs,
Device Architecture, arquite
tivos de cómputo), la arquit
ralela propuesta por NVIDI
de GPUs durante los últimos
entorno de programación m
profesional, obteniendo ade
Por otra parte, hay program
código CUDA es más inteli
dagógico) que el código O
drían influir en la decisión d
ñar CUDA en lugar de Oper

En relación a la parte pr
CUDA, una cuestión import
GPUs CUDA en el laborato
te, principalmente desde un punto de vista económico, pero teniendo en cuenta al mismo tiempo la calidad del aprendizaje. Por un lado, cuando estamos montando un laboratorio CUDA, instalar GPUs CUDA en todos los ordenadores del laboratorio podría no ser asequible en términos del coste económico que conlleva esta estrategia, dado el precio de estas tarjetas. Por otro lado, el enfoque contrario consiste en pedir a los estudiantes que inicien sesión en un servidor remoto con GPU, el cuál podría estar bien en el propio laboratorio, bien en otra sala de la universidad. Sin embargo, esta opción podría derivar en unas malas condiciones de aprendizaje, como se verá más adelante. Una solución intermedia donde se usara un sistema de colas de trabajo tampoco resulta eficaz, como se explicará después.

En este artículo proponemos una solución eficiente para dotar de GPUs un laboratorio docente. Nuestra propuesta se basa en usar rCUDA [8, 9] (CUDA remoto), un *middleware* que permite a programas ejecutándose en un ordenador utilizar GPUs ubicadas en servidores remotos de forma concurrente y transparente. De esta manera, los estudiantes pueden compartir la GPU de un servidor desde sus puestos de trabajo en el laboratorio sin necesidad de iniciar sesión en dicho servidor, evitando de esta forma los problemas anteriores al mismo tiempo que se mantiene la calidad del aprendizaje, tal y como se detallará en secciones posteriores.

El resto del artículo está organizado de la siguiente manera. En la Sección 2 presentamos el *middleware* rCUDA. En la Sección 3 presentamos en detalle nuestra propuesta, comparándola con las tres formas alternativas mencionadas anteriormente. También se muestran resultados de un escenario real para demostrar que nuestra propuesta es factible. Finalmente, la Sección 4 resume las principales conclusiones de este trabajo.

[1]http://www.green500.org

[2]NVIDIA, CUDA API Reference Manual 6.5, 2014.

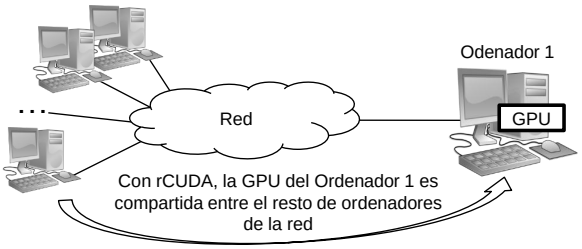

Figura 1: Escenario de ejemplo con rCUDA.

2. rCUDA: CUDA remoto

Tal y como hemos adelantado anteriormente, CUDA es una tecnología creada por NVIDIA que proporciona una plataforma de computación paralela y un modelo de programación para ser usado con GPUs NVIDIA o compatibles. CUDA aprovecha el gran poder de cómputo de las GPUs para acelerar determinadas partes de las aplicaciones, reduciendo así su tiempo de ejecución. No obstante, es el programador quién decide qué partes de la aplicación se ejecutan en la CPU tradicional, y qué partes son ejecutadas en la GPU. Dicha decisión depende, básicamente, del nivel de paralelización que se puede conseguir para las diferentes partes de la aplicación.

rCUDA [8, 9] (CUDA remoto) es un *middleware* que permite compartir dispositivos remotos compatibles con CUDA de un modo totalmente transparente al programador. De esta manera, una GPU instalada en un ordenador de una red (el servidor proporcionando servicios de GPU) puede ser utilizada de forma concurrente por otros ordenadores de la red (los clientes que requieren servicios de GPU) para acelerar aplicaciones CUDA, tal y como muestra la Figura 1. rCUDA proporciona a las aplicaciones, de forma transparente, acceso a GPUs instaladas en ordenadores remotos, de manera que éstas no son conscientes de estar utilizando un dispositivo remoto.

La Figura 2 muestra la arquitectura de rCUDA. Como podemos ver, se trata de una arquitectura distribuida cliente-servidor. Cuando una aplicación requiere los servicios de la GPU, el cliente rCUDA redirige la petición al servidor a través de la red. Obsérvese que la aplicación continúa utilizando la misma interfaz que un programa ordinario CUDA (i.e., la API original de CUDA). Así pues, no es necesario modificar la aplicación. La forma de conseguir esto es mediante la sustitución en tiempo de ejecución de la librería de CUDA

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

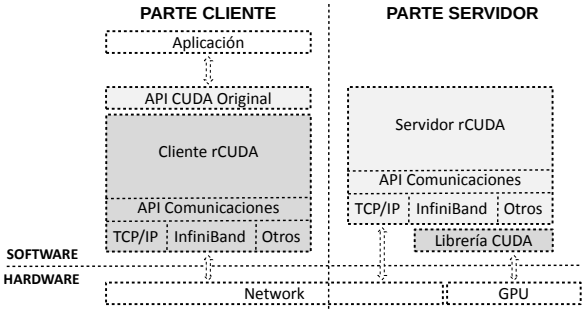

Figura 2: Arquitectura de rCUDA.

original por la de rCUDA, que presenta la misma interfaz, tal y como hemos comentado. De esta manera, cuando una aplicación llama a una función CUDA, realmente se ejecuta la función correspondiente de la librería de rCUDA, la cual redirige la llamada al servidor remoto.

Una vez el servidor de rCUDA recibe la petición del cliente, ésta es ejecutada en la GPU real. Cuando finaliza la tarea en la GPU, el servidor rCUDA reenvía la respuesta al cliente rCUDA y, finalmente, el cliente rCUDA entrega el resultado a la aplicación que inicialmente realizó la petición. Obsérvese que la aplicación no es consciente de haber accedido a una GPU remota, sino que todo el proceso es automático y transparente a la aplicación.

La comunicación entre el cliente de rCUDA y el servidor se realiza a través de un protocolo de comunicaciones propietario que utiliza la red disponible entre el ordenador donde se ejecuta la aplicación y el ordenador donde se encuentra la GPU. Actualmente, rCUDA proporciona dos implementaciones diferentes de este protocolo de comunicaciones: (1) una optimizada para redes InfiniBand, que utiliza InfiniBand Verbs y que está dirigida a clusters de altas prestaciones; y (2) una versión genérica que utiliza sockets TCP, compatible con la gran mayoría de redes, y que va dirigida a los entornos en los cuales el rendimiento no es tan importante, como por ejemplo los laboratorios docentes. En este último escenario, es posible utilizar la red Ethernet disponible en los propios laboratorios, no siendo necesario así ningún coste adicional.

La última versión de rCUDA, disponible en `http://rcuda.net/`, soporta la Runtime API[3] y la Driver API[4] de la versión 6.5 de CUDA. También ofrece soporte para la mayoría de las rutinas de las librerías CUDA más comunes, como son cuBLAS[5], cuFFT[6], cuRAND[7] y cuSPARSE[8]. Además, rCUDA se distribuye de forma gratuita, por lo que es posible utilizar dicha tecnología sin ningún coste adicional.

A continuación detallamos los pasos a llevar a cabo para comenzar a utilizar rCUDA en un laboratorio docente. En primer lugar, copiaremos en el ordenador sin GPU (el puesto de trabajo del alumno) el fichero que contiene la librería cliente de rCUDA, que sustituye a la librería de CUDA original. La librería de rCUDA será la encargada de interceptar las llamadas CUDA de los programas que creen los alumnos y reenviarlas al servidor de rCUDA. En segundo lugar, iniciaremos el servidor de rCUDA en el ordenador servidor con GPU. El servidor de rCUDA es un proceso demonio que se ejecuta en segundo plano y que permanece a la escucha de peticiones en un puerto TCP determinado. Como puede apreciarse, el proceso de instalación de rCUDA es tremendamente sencillo y no requiere de conocimientos especiales.

Una vez tenemos la librería rCUDA instalada en el puesto de trabajo del alumno y el demonio rCUDA arrancado en el servidor con GPU, ejecutaremos las aplicaciones CUDA de la forma habitual. rCUDA, de forma transparente al alumno, interceptará cada llamada CUDA y la hará llegar al servidor. Este último la ejecutará en la GPU que tiene instalada y, cuando finalice la llamada, retornará el resultado al cliente rCUDA, el cual lo hará llegar a la aplicación que realmente realizó la llamada CUDA.

Nótese que no es necesario que los alumnos realicen ninguna acción adicional a las que realizan cuando ejecutan un programa con CUDA. El servidor de rCUDA es un demonio que, una vez instalado por los administradores del laboratorio, se ejecutará en segundo plano en el ordenador servidor de forma permanente. Mientras que la librería de rCUDA instalada en los puestos de trabajo de los estudiantes tiene el mismo nombre que la librería CUDA original, por lo que éstos no serán conscientes de estar utilizando una GPU remota.

3. Equipando los Laboratorios Docentes con GPUs

Tal y como hemos comentado en la sección de introducción, a la hora de equipar un laboratorio docente con GPUs existen diversas opciones. En esta sección profundizamos en las diferentes formas de dotar de GPUs los laboratorios docentes, comparando sus costes y estudiando las ventajas e inconvenientes que presentan cada una de ellas. También presentamos nuestra propuesta de uso de rCUDA, comparándola con el resto de opciones. No obstante, antes de abordar las diferentes formas de equipar un laboratorio docente con GPUs, mostramos la importancia de seleccionar un modelo de GPU adecuado a los fines docentes.

[3]NVIDIA, CUDA API Reference Manual 6.5, 2014.
[4]NVIDIA, CUDA Driver API 6.5, 2014.
[5]NVIDIA, CUBLAS Library 6.5, 2014.
[6]NVIDIA, CUFFT Library 6.5, 2014.
[7]NVIDIA, CURAND Library 6.5, 2014.
[8]NVIDIA, CUSPARSE Library 6.5, 2014.

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

Tarjeta Gráfica	Coste
NVIDIA GeForce GT 520[10]	100€
NVIDIA GeForce GTX 590[11]	300€
NVIDIA GeForce GTX 780 Ti[12]	600€

Cuadro 1: Comparación de GPUs NVIDIA para ordenadores de sobremesa.

3.1. La importancia de seleccionar la GPU adecuada

Desde nuestro punto de vista, es importante que los estudiantes conozcan de los beneficios que proporciona la computación paralela con GPUs, entre ellos la importante reducción en el tiempo de ejecución, con el objetivo de motivarles para que hagan el esfuerzo de aprender un nuevo paradigma de programación. Así pues, las GPUs utilizadas en el laboratorio deben superar claramente las prestaciones de las CPUs para que realmente se aprecien los beneficios de las GPUs. De lo contrario, los estudiantes podrían no estar motivados y no sacarían todo el provecho posible del tiempo empleado en el laboratorio. En esta sección mostramos la importancia de escoger un modelo de GPU adecuado con el fin de estimular a los estudiantes y tener, en consecuencia, una buena calidad de aprendizaje.

El Cuadro 1 muestra los precios de tres GPUs NVIDIA para ordenadores de sobremesa disponibles actualmente en el mercado[9]. Como podemos ver, el coste de las GPUs varía en un rango muy amplio, el cual depende principalmente de la capacidad de cálculo de las mismas, así como de la cantidad de memoria que incorporan. Aunque el Cuadro 1 compara las tres GPUs desde el punto de vista económico, también es necesario considerar su poder computacional, dado que una GPU con poca potencia podría desmotivar a los alumnos. A continuación mostramos un sencillo experimento comparando estas tres GPUs.

La Figura 3 presenta los resultados del programa *matrixMul* extraído del paquete NVIDIA CUDA Samples 6.5[13]. Este paquete, distribuido junto con CUDA, contiene una serie de programas de ejemplo comúnmente utilizados durante el aprendizaje de CUDA. El programa *matrixMul* realiza una multiplicación de matrices en la GPU. Este programa ha sido seleccionado porque ilustra varios principios de la programación paralela con CUDA y, en nuestra opinión, es un buen punto de partida para motivar a los asistentes a un curso de CUDA. Hemos ejecutado el programa utilizando las 3 GPUs detalladas en el Cuadro 1. Además, también incluimos a modo de referencia los resultados de una multiplicación de matrices similar realizada sin GPU, utilizando la popular librería de álgebra lineal GotoBLAS2 [10] sobre una CPU moderna.

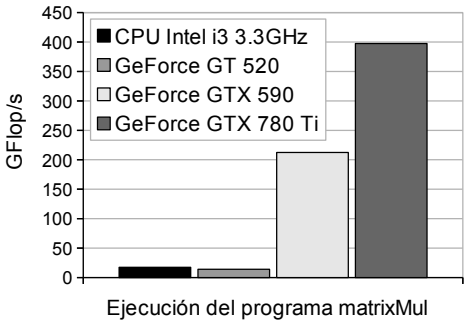

Figura 3: Ejecución del programa *matrixMul* con 3 GPUs diferentes y comparado, a su vez, con una multiplicación de matrices similar ejecutada en una CPU utilizando la librería GotoBLAS2. Se utiliza arimética de simple precisión.

Como puede observarse en la Figura 3, utilizar una GPU económica como la GeForce GT 520 podría no ser suficiente para motivar a los estudiantes a aprender un nuevo paradigma de programación, dado que pueden obtenerse mejores resultados utilizando una CPU de coste medio. Utilizar una GPU de este estilo podría incluso desmotivar a los estudiantes, dificultando su atención durante el resto del curso de CUDA. En cambio, si utilizamos GPUs más avanzadas, como por ejemplo la GeForce GTX 590 o la GTX 780 Ti mostradas en el Cuadro 1, el beneficio de la computación paralela con GPUs resulta obvio. De esta manera, esta tecnología de aceleración resulta más atractiva a los estudiantes y la experiencia docente sería, en consecuencia, mejor. En cualquier caso, desde nuestro punto de vista, cuanto mejores sean las prestaciones de la GPU, mejor será la experiencia docente. Por dicho motivo, en las siguientes secciones utilizaremos el modelo de GPU GTX 780 Ti.

A continuación mostramos cuatro maneras diferentes de equipar un laboratorio docente con GPUs (incluyendo nuestra propuesta de rCUDA) comparando sus ventajas y sus puntos débiles.

3.2. Instalar GPUs CUDA en todos los ordenadores del laboratorio

Instalar GPUs CUDA en todos los ordenadores del laboratorio sería la opción más deseable en términos de

[9] Aunque los centros de datos usan GPUs de gama alta, como pueden ser los modelos Tesla K20 o Tesla K40, con un precio aproximado de 2.000 y 4.000 euros por unidad, respectivamente, a la hora de equipar un laboratorio docente se pueden emplear otros modelos notablemente más baratos.

[10] http://www.geforce.com/hardware/desktop-gpus/geforce-gt-520/specifications, 2014.

[11] http://www.geforce.com/hardware/desktop- gpus/geforce-gtx-590/specifications, 2014.

[12] http://www.geforce.com/hardware/desktop- gpus/geforce-gtx-780-ti/specifications, 2014.

[13] NVIDIA, CUDA Samples Reference Manual 6.5, 2014.

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

Tarjeta Gráfica	Coste
NVIDIA GeForce GTX 780 Ti	20x600€
TOTAL	12.000€

Cuadro 2: Coste económico de equipar un laboratorio compuesto por 20 ordenadores con GPUs.

prestaciones y calidad de aprendizaje. Por esta razón, usaremos este enfoque como referencia a la hora de comparar con el resto de propuestas.

En el Cuadro 2 presentamos el coste económico para equipar un laboratorio compuesto por 20 ordenadores, cada uno con una NVIDIA GeForce GTX 780 Ti. Para los ordenadores de los puestos de trabajo de los alumnos se prevé una configuración basada en Intel Core i3-3220 a 3,3GHz y 4GB de memoria RAM, tal y como se ha sugerido en la sección anterior.

Obsérvese que en el Cuadro 2 hemos omitido el coste de los 20 ordenadores y de la red porque asumimos que para equipar un laboratorio con GPUs se partiría de una configuración donde ya se dispone de una red Ethernet y de los puestos de trabajo de los alumnos, y por lo tanto el coste de dicho equipamiento sería un valor constante, no modificando así el coste relativo de las diferentes propuestas mostradas en este trabajo.

3.3. Usar un servidor remoto con GPU

En un primer intento de reducir el coste del laboratorio CUDA, una posible solución sería disponer de una única GPU en un servidor, de manera que los estudiantes iniciaran sesión en este servidor remoto desde sus puestos de trabajo en el laboratorio. Con este enfoque, evitaríamos la instalación de una GPU en cada ordenador, y el coste total del laboratorio sería similar al que mostramos en el Cuadro 3. En este cuadro se presenta el coste de equipar un laboratorio con un servidor adicional con una GPU NVIDIA GeForce GTX 780 Ti. Obsérvese que la configuración del servidor es mucho mejor que la configuración típicamente usada para los ordenadores de los estudiantes, puesto que el servidor tendrá que albergar los entornos gráficos de los estudiantes, así como sus herramientas de programación.

Como podemos observar, el coste total del laboratorio disminuye notablemente. No obstante, esta propuesta podría resultar en una mala experiencia docente debido a la sobrecarga que conlleva en el servidor: todos los estudiantes iniciando sesiones gráficas en el servidor para utilizar los entornos gráficos de programación, todos los estudiantes consumiendo la memoria principal del servidor, la sobrecarga adicional de la CPU del servidor al compilar y ejecutar los programas, etc. Seguidamente presentamos experimentos utilizando un laboratorio similar al del Cuadro 3 con el fin de

Tarjeta Gráfica	Coste
Intel Core i7-4790 3,6GHz 32GB RAM	1x1000€
NVIDIA GeForce GTX 780 Ti	1x600€
TOTAL	1.600€

Cuadro 3: Coste de equipar un laboratorio con 1 servidor remoto con GPU.

mostrar esta sobrecarga.

Por ejemplo, la Figura 4 muestra cómo el tiempo de compilación del programa matrixMul, utilizado en secciones anteriores, aumenta de forma proporcional al número de usuarios que compilan el mismo programa de forma simultánea. Los resultados de esta figura muestran el tiempo medio de compilación para cada usuario. Como podemos observar, el tiempo se incrementa de los 10 segundos iniciales cuando sólo hay un alumno compilando, a los 40 segundos cuando los 20 estudiantes están compilando el programa al mismo tiempo. Teniendo en cuenta que los estudiantes están aprendiendo, las compilaciones van a ser muy frecuentes y este tiempo de espera seguro que empeora la calidad del aprendizaje. Obsérvese además que los procesadores de los ordenadores de los estudiantes permanecen poco utilizados, no amortizando así su coste.

Otra desventaja del uso de un único servidor es la disminución de las prestaciones cuando varios estudiantes están ejecutando programas que usan la GPU. Por ejemplo, en la Figura 5 presentamos el rendimiento medio del programa matrixMul cuando varios alumnos lo están ejecutando al mismo tiempo. Nuevamente, los resultados empeoran considerablemente si los comparamos con la ejecución de una única instancia del programa. De esta manera, el rendimiento de la GPU se reduce casi en un 80 % cuando 5 estudiantes ejecutan el programa de forma simultánea, en comparación con

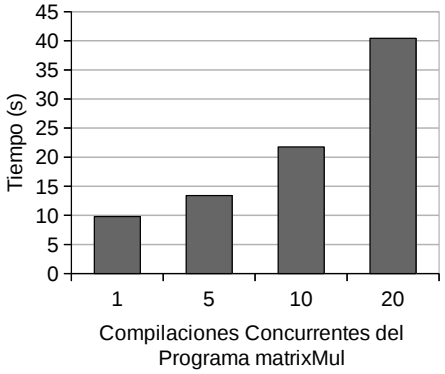

Figura 4: Tiempo medio de compilación del programa matrixMul cuando uno o más usuarios han iniciado sesión en el servidor remoto con GPU y están compilando el programa al mismo tiempo.

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

el rendimiento que se obtiene cuando un único estudiante utiliza la GPU. Este bajo rendimiento también va a deteriorar la calidad del aprendizaje.

En cualquier caso, que todos los estudiantes compilen al mismo tiempo, o que usen la GPU a la vez, sería el caso peor, dado que en general no se da este nivel de concurrencia y sincronización en los laboratorios. Una situación más razonable sería, por ejemplo, que 5 de los 20 alumnos compilaran al mismo tiempo. Del mismo modo, es más razonable esperar que 5 de los 20 alumnos estén usando la GPU a la vez.

3.4. Utilizar un sistema de colas

Una solución alternativa es instalar un sistema de colas de ejecución en el laboratorio, de manera que los programas se escriben y compilan en los puestos de trabajo de los alumnos y para su ejecución se lanzan a la cola. El servidor con GPU iría extrayendo los trabajos de la cola y los ejecutaría uno tras otro en la GPU. No obstante, esta opción presenta un primer problema, que es la sobrecarga de instalar y administrar el sistema de colas. Alternativamente se podría prescindir del sistema de colas y pedir a los alumnos que se conectaran al servidor para ejecutar el programa, lo cual produciría los mismos tiempos de ejecución ya vistos en la sección anterior.

En cualquier caso, estas dos opciones presentan el inconveniente de que el alumno debe ejecutar el programa de forma manual, no pudiendo sacar partido de las facilidades ofrecidas por entorno gráfico de programación, con lo que la calidad del aprendizaje se vería seriamente disminuida.

3.5. Utilizar rCUDA

El enfoque presentado en las dos secciones anteriores disminuye considerablemente el coste del laboratorio docente, pero también afecta a la calidad del aprendizaje. En esta sección proponemos una solución con el mismo coste que el mostrado en el Cuadro 3, pero manteniendo la experiencia docente del enfoque presentado en la Sección 3.2.

Nuestra propuesta se basa en el uso del *middleware* rCUDA. Como hemos explicado en la Sección 2, rCUDA permite a programas ejecutándose en un ordenador utilizar GPUs ubicadas en servidores remotos de forma concurrente. De esta manera, los estudiantes pueden compartir, de forma transparente y concurrente, una misma GPU remota desde sus puestos de trabajo en el laboratorio, sin necesidad de iniciar sesión en un servidor remoto. Así pues, los estudiantes utilizan sus puestos de trabajo para cargar el entorno gráfico de programación y para desarrollar y compilar sus programas. De ese modo, el servidor que ofrece los servicios de la GPU no se sobrecarga con dichas

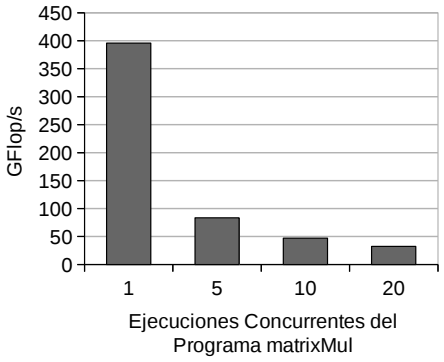

Figura 5: Rendimiento medio del programa matrixMul cuando uno o más alumnos han iniciado sesión en el servidor remoto con GPU y están ejecutando el programa al mismo tiempo.

tareas. Además, los ejercicios implementados durante la sesión de laboratorio se ejecutan en los ordenadores de los estudiantes y rCUDA, de forma transparente, ejecuta la parte de dichos programas que requiere GPU en el servidor remoto, mientras que la parte que no requiere GPU se ejecuta en los puestos de trabajo de los estudiantes. Esto permite que el servidor tampoco se sobrecargue con las partes de los programas de los estudiantes que no requieren el uso de la GPU. Adicionalmente, rCUDA es completamente compatible con CUDA, por lo que los programas no necesitan ser modificados y los alumnos aprenden únicamente CUDA sin necesidad de preocuparse por rCUDA, que es transparente a los estudiantes.

En la Figura 6 mostramos el mismo experimento de la Figura 5, con el mismo equipamiento que el mostrado en el Cuadro 3, pero utilizando rCUDA en lugar de iniciar sesión en el servidor remoto. Como podemos ver, en este caso las prestaciones de usar una GPU remota (las barras etiquetadas como 'rCUDA' en la figura) también son inferiores, un 10 % menos cuando 5 estudiantes ejecutan el programa de forma simultánea, en comparación con el rendimiento que proporciona una GPU local (la línea negra etiquetada como 'CUDA' en la figura). No obstante, dicho rendimiento sigue siendo claramente superior al proporcionado por una CPU (véase la Figura 3). De este modo, el coste del laboratorio ha sido notablemente reducido pero se mantiene la calidad del aprendizaje.

A continuación, presentamos varios experimentos adicionales con el fin de demostrar que el uso de rCUDA en laboratorios docentes es factible y no supone una reducción en la calidad del aprendizaje. Para ello, hemos utilizado programas que pueden encontrase en el paquete NVIDIA CUDA Samples 6.5.

Los programas que hemos seleccionado son códigos que pueden ser muy útiles en las prácticas de CUDA, dada su fácil comprensión y enfoque didácti-

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

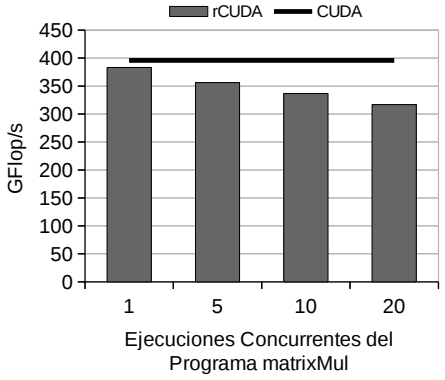

Figura 6: Rendimiento medio del programa matrixMul cuando uno o más alumnos están usando el servidor remoto con GPU a través de rCUDA y están ejecutando el programa al mismo tiempo.

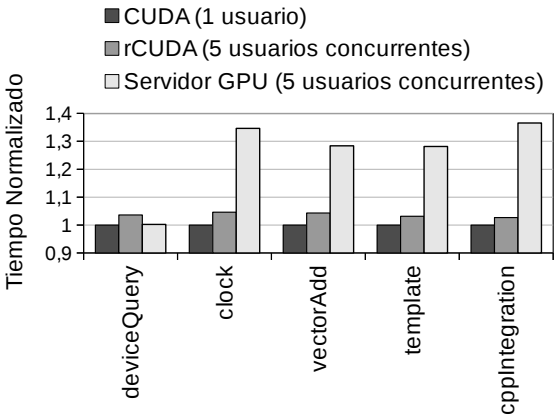

Figura 7: Tiempo de ejecución normalizado de varios programas extraídos del paquete NVIDIA CUDA Samples con CUDA (el programa es ejecutado por un único alumno), con rCUDA (el programa es ejecutado simultáneamente por 5 alumnos diferentes que comparten la misma GPU remota), y utilizando un servidor con GPU (5 estudiantes diferentes inician sesión en el servidor y ejecutan el programa de forma simultánea).

co. En concreto, los programas utilizados han sido los siguientes:

- *deviceQuery*: es un programa muy sencillo que únicamente consulta las propiedades de la GPU (capacidad de cómputo, memoria, número de cores...). Puede ser muy útil en sistemas con varias GPUs, para seleccionar la GPU más potente.
- *clock*: este ejemplo muestra cómo utilizar la función *clock* para medir las prestaciones de una función ejecutada en la GPU. Además, muestra la nomenclatura utilizada para ejecutar funciones en la GPU (referidas con el término *kernel* en el ámbito de CUDA).
- *vectorAdd*: un programa que implementa una suma de vectores elemento a elemento. Además de mostrar un kernel con funcionalidad, también introduce el tratamiento de errores con CUDA.
- *template*: una plantilla simple que puede ser posteriormente utilizada como punto de partida para que los estudiantes realicen un nuevo proyecto con CUDA.
- *cppIntegration*: este programa muestra cómo integrar CUDA en una aplicación C++ existente.

En la Figura 7 mostramos el tiempo necesario para ejecutar los anteriores programas. El equipamiento utilizado para estos experimentos ha sido el que aparece en los Cuadros 2 y 3 de las secciones anteriores. En el caso de CUDA, se ha usado el equipamiento del Cuadro 2, dado que este caso corresponde al escenario presentado en la Sección 3.2, donde cada alumno dispone de un puesto de trabajo con GPU. En el caso de rCUDA, se ha utilizado el equipamiento del Cuadro 3, tal y como hemos descrito al inicio de esta misma sección. Finalmente, en el caso del servidor con GPU, el equipamiento empleado es también el del Cuadro 3, usado de la forma descrita en la Sección 3.3.

En la Figura 7 mostramos el tiempo total de ejecución normalizado de los programas cuando son ejecutados en los diferentes escenarios. Se ha considerado un nivel de concurrencia de 5 alumnos simultáneos ejecutando el mismo programa.

Tal y como podemos observar, el sobrecoste de utilizar rCUDA es inferior al 5 %, mientras que la opción de iniciar sesión en un servidor con GPU introduce, en general, una sobrecarga entorno al 30 %. La única excepción se produce con el programa *deviceQuery*, para el cual compartir un servidor con GPU no añade prácticamente sobrecoste con respecto a utilizar los propios puestos de trabajo con GPUs. La razón la encontramos analizando las aplicaciones en mayor detalle, siendo *deviceQuery* la única de todas ellas que no realiza cómputo en la GPU (i.e., no ejecuta ningún kernel). Dado que lo habitual será que las aplicaciones realicen cálculos en la GPU, entendemos que estos experimentos demuestran que el uso de rCUDA en laboratorios docentes es una posibilidad a tener en cuenta a la hora de crear un laboratorio docente para CUDA, mejorando en gran medida los resultados obtenidos con respecto a utilizar un servidor con GPU, pero sin incrementar el coste económico.

4. Conclusiones

En este artículo hemos propuesto una solución eficiente para equipar con GPUs los laboratorios docentes. Nuestra propuesta está basada en el uso de rCUDA (CUDA remoto), un *middleware* que permite a programas ejecutándose en un ordenador utilizar GPUs ubicadas en servidores remotos de forma concurrente. De

Andorra La Vella, del 8 al 10 de julio 2015
ISBN: 978-99920-70-10-9

esta manera, los estudiantes pueden compartir, de forma transparente y concurrente, una misma GPU remota desde sus puestos de trabajo en el laboratorio, sin necesidad de iniciar sesión en un servidor con GPU.

Adicionalmente, hemos comparado nuestra propuesta con otras tres maneras más directas e inmediatas de equipar un laboratorio docente: (1) instalando GPUs en todos los ordenadores, (2) accediendo a un servidor remoto con GPU y (3) añadiendo un servidor con GPU e instalando un sistema de colas de trabajo. Para comparar los cuatro enfoques considerados, hemos mostrado resultados de un escenario real: los experimentos han sido realizados en un laboratorio docente con 20 ordenadores. En el caso de las propuestas 2, 3 y también de la nuestra propia, se ha utilizado además un servidor con GPU.

Nuestro estudio demuestra que el planteamiento consistente en instalar GPUs en todos los ordenadores proporciona la mejor calidad de aprendizaje posible, pero el coste que conlleva podría no ser asequible para algunas instituciones docentes. Con el objetivo de reducir dicho coste, la estrategia de no tener GPUs en los puestos de trabajo, sino iniciar sesión en un servidor remoto con GPU, reduce los gastos a más de la mitad. Sin embargo, esta metodología también deteriora la calidad del aprendizaje. Finalmente, la propuesta basada en el uso de rCUDA consigue ambos objetivos: el coste del laboratorio se reduce notablemente mientras la calidad del aprendizaje se mantienen.

Nótese, además, que nuestra propuesta es totalmente compatible con la tendencia actual de proporcionar máquinas virtuales a los alumnos para que puedan completar sus trabajos prácticos desde fuera del laboratorio docente. En este sentido, dentro de la máquina virtual se introduce la parte cliente de rCUDA, que daría acceso a la GPU instalada en el servidor con GPU. De hecho, dado que proporcionar acceso a una GPU a los programas que se ejecutan dentro de una máquina virtual es difícil, nuestra propuesta para el uso de rCUDA es una forma más sencilla de proporcionar dicho acceso.

Como trabajo futuro se plantea poner en marcha esta propuesta en una asignatura de programación paralela.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Escola Tècnica Superior d'Enginyeria Informàtica de la Universitat Politècnica de València y también por el Departament d'Informàtica de Sistemes i Computadors (DISCA) de dicha universidad.

Referencias

[1] Haicheng Wu, Gregory Diamos, Tim Sheard, Molham Aref, Sean Baxter, Michael Garland, y Sudhakar Yalamanchili, "Red Fox: An Execution Environment for Relational Query Processing on GPUs," en *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*, 2014.

[2] Daniel P. Playne y Kenneth A. Hawick, "Data Parallel Three-Dimensional Cahn-Hilliard Field Equation Simulation on GPUs with CUDA," en *Proceedings of 2009 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2009.

[3] Yuancheng Luo y R. Duraiswami, "Canny edge detection on NVIDIA CUDA," en *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

[4] Abhijeet Gaikwad y Ioane Muni Toke, "GPU Based Sparse Grid Technique for Solving Multidimensional Options Pricing PDEs," en *Proceedings of the 2nd Workshop on High Performance Computational Finance*, 2009.

[5] Ichitaro Yamazaki, Tingxing Dong, Raffaele Solcà, Stanimire Tomov, Jack Dongarra, y Thomas Schulthess, "Tridiagonalization of a dense symmetric matrix on multiple GPUs and its application to symmetric eigenvalue problems," *Concurrency and Computation: Practice and Experience*, vol. 26, núm. 16, 2014.

[6] Kyle E. Niemeyer y Chih-Jen Sung, "Recent Progress and Challenges in Exploiting Graphics Processors in Computational Fluid Dynamics," *J. Supercomput.*, vol. 67, núm. 2, 2014.

[7] Khronos OpenCL Working Group, *OpenCL 2.0 Specification*, 2013.

[8] Carlos Reaño, Rafael Mayo, Enrique S. Quintana-Ortí, Federico Silla, José Duato, y Antonio J. Peña, "Influence of InfiniBand FDR on the performance of remote GPU virtualization," en *IEEE International Conference on Cluster Computing*, 2013.

[9] Antonio J. Peña, Carlos Reaño, Federico Silla, Rafael Mayo, Enrique S. Quintana-Ortí, y José Duato, "A complete and efficient CUDA-sharing solution for HPC clusters," *Parallel Computing*, vol. 40, núm. 10, 2014.

[10] Texas Advanced Computing Center (TACC), University of Texas (Austin), "GotoBLAS2 Library 1.13," https://www.tacc.utexas.edu/research-development/tacc-software/gotoblas2, 2014.