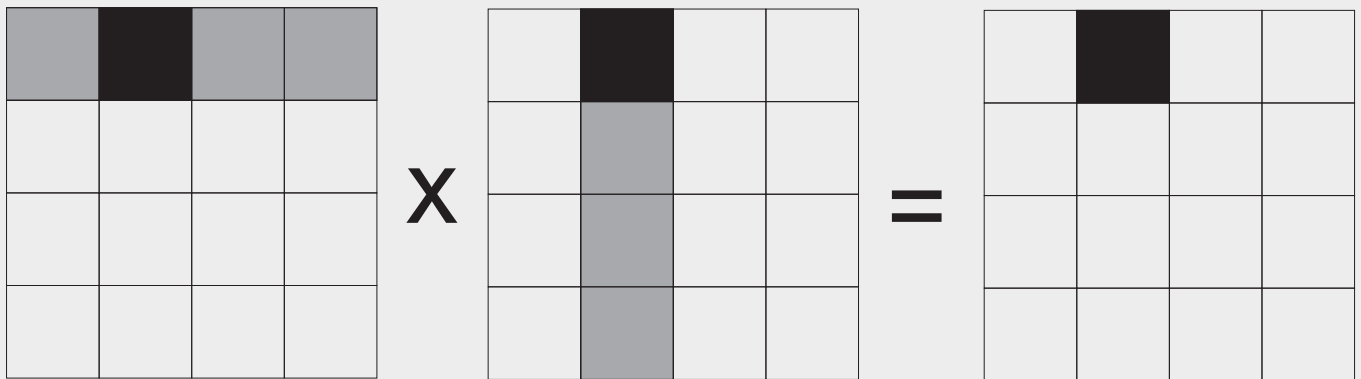


Práctica 7

Multiplicación de matrices y reducciones en CUDA



47	2	3	57	5	12	7	8
10	20	6	13	14	15	16	17
19	11	21	22	23	68	25	26
38	29	64	31	32	33	35	34
37	28	39	49	53	42	41	52
46	1	48	40	61	51	44	43
55	71	4	58	69	62	50	60
30	65	66	67	24	59	70	56



47	57	15	17
38	64	68	35
46	49	61	52
71	67	69	70

**Máster Oficial en
Informática Gráfica,
Juegos y
Realidad Virtual**

**Procesadores
Gráficos**



Universidad
Rey Juan Carlos

*"Aún tiene que probarse que la inteligencia
tenga algún valor para la supervivencia."*

-- Arthur C. Clarke

*"Programar de forma altamente paralela es muy sencillo,
siempre y cuando no te importe en absoluto el rendimiento"*

--Wen Mei Hwu

Introducción

El objetivo de esta práctica consiste en reforzar los conceptos de arquitectura que se han introducido en el segundo bloque de la asignatura de Procesadores Gráficos, para lograrlo, aplicaremos estos conocimientos sobre la base práctica del diseño y la programación de algoritmos muy comunes como pueden ser la multiplicación de matrices y la reducción de vectores en GPGPU.

La presente práctica consta de dos partes relativas a los dos algoritmos que vamos a implementar a partir de un código base. Al igual que en prácticas anteriores, el guión de esta práctica se complementa con una serie de lecturas que al alumno se recomienda leer antes del día de la práctica, para poder aprovechar el tiempo en el laboratorio al máximo aplicando los conceptos contemplados en esos textos.

En este caso, se recomienda echar un vistazo al manual de programación de CUDA, que podeis encontrar aquí:

http://www.nvidia.com/object/cuda_develop.html

Los ejemplos de algoritmos que vamos a tratar en esta práctica son tan sencillos que CUDA trae librerías que los realizan de forma mucho más eficiente, pero hemos decidido escogerlos para no complicar excesivamente los primeros pasos a dar con esta nueva filosofía de programación en la arquitectura que estamos estudiando en clase. De nuevo, con el objetivo de hacer una introducción lo más asequible posible, en esta práctica no nos preocuparemos de temas de rendimiento, y la implementación a realizar se refinará en posteriores prácticas.

IMPORTANTE:

- En esta práctica habrá que entregar una memoria escrita en la que se respondan claramente a las preguntas y cuestiones propuestas en cada apartado.
- Es necesario incluir el código fuente o, al menos, indicar las modificaciones realizadas sobre el código existente.
- Será absolutamente **obligatorio** comentar los resultados obtenidos.

Normativa del laboratorio

- El criterio más importante es la funcionalidad: un programa que funciona siempre tiene más posibilidades de llevarse una buena puntuación; no se valorarán aquellos programas que no funcionen. Una práctica o proyecto modesto será evaluada mucho más favorablemente que un "proyecto" ambicioso que sólo da *core-dumps*. Los

siguientes criterios que se tendrán en cuenta (y que hay que cuidar al realizar las prácticas) son:

- La manera de resolver el problema con el programa
 - Estructuras de datos y diseño de los algoritmos
 - Claridad y documentación en el código
 - Eficiencia y elegancia en la implementación.
-
- ¡Por favor, no hagáis trampas! Se procura alentar el diálogo y el trabajo en equipo, pero por favor trabajad de forma independiente (a menos que el trabajo sea en grupos). Trabajos muy similares serán considerados como copias, a menos que la naturaleza de lo pedido sea tan restrictiva que justifique las similitudes. Y una copia implica el suspenso automático. Simplemente piénsalo de esta manera: hacer trampas dificulta el aprendizaje y la diversión de conseguir hacerlo. Es vuestra responsabilidad proteger vuestro trabajo y asegurarnos que no se convierte en el de otro.
 - Si se utiliza (o mejora) código fuente u otro material obtenido a través de internet, la biblioteca... debe darse el crédito a los autores y pedir permiso de ser necesario (si tiene una licencia restrictiva). Tomar código de un libro o de internet sin tener en cuenta estas consideraciones será considerado copia.
 - Está terminantemente prohibido la práctica de técnicas de *overclocking* en las tarjetas gráficas del laboratorio, así como desbloquear los procesadores de vértices y fragmentos de los chips gráficos. Este tipo de acciones pueden dañar físicamente el equipo del laboratorio y los alumnos responsables serán amonestados severamente.
 - Las prácticas (código y memoria explicativa) deberán entregarse en los plazos indicados. Un retraso en la entrega supondrá una reducción en la nota de dicha práctica de según el siguiente baremo: 1 día tarde: -10%, 3 días tarde: -20%, 1 semana tarde: -40%

Recomendaciones

En principio, la práctica se puede hacer en parejas, si alguien no encuentra pareja o, por problemas de horario tiene complicado el poder compaginarlo con el trabajo, también es posible hacerla de forma individual, esta práctica es sencilla y no debería suponer una dificultad añadida el no tener compañero/a para realizarla.

Si algún alumno no tuviera tiempo para terminarla en el tiempo de las tutorías grupales de la asignatura y no dispone del hardware gráfico necesario para realizarla tiene dos opciones:

- Puede realizarla cuando le resulte más cómodo en el aula de libre acceso S09 del edificio de Laboratorios II (Campus de Móstoles) que dispone de 15 ordenadores con tarjeta GeForce7900. Indícale al responsable del aula que necesitáis un ordenador con este hardware para realizar la práctica y os dirá el que podéis utilizar.
- Utilizar las opciones de emulación de CUDA en cualquier ordenador con Windows y Visual Studio, o Linux con el paquete que podeis encontrar en: http://www.nvidia.com/object/cuda_get.html

1. Notas sobre multiplicación de matrices

1. Descomprimir el fichero practica7_mult_matrices.rar en <Dir_proyecto>\NVIDIA CUDA SDK\projects (o lo más parecido posible -ya que en los ordenadores del laboratorio ese directorio está protegido- para evitar problemas con las dependencias de ficheros)
2. Editar las funciones MatrixMulDevice(...) en matrixmul.cu y MatrixMulKernel() en matrixmul_kernel.cu para completar la funcionalidad que falta en el device. No es necesario cambiar el código en otras partes de los ficheros fuente (aunque conviene leerlos para entender su funcionamiento).
El tamaño de la matriz está definida tal manera que un solo bloque de hebras sea suficiente para calcular la solución de toda la matriz.
3. Tenemos varios modos de operación para la aplicación.
 - a. Sin argumentos:
La aplicación creará un par de matrices inicializadas de forma aleatoria. Una vez la multiplicación en el device es invocada, calcula la solución también en la CPU para poder comparar ambas aproximaciones. Si coincide (dentro de una cierta tolerancia), imprimirá “Test PASSED” por pantalla antes de terminar.
 - b. Un argumento:
La aplicación se comporta de forma parecida a la anterior, al multiplicar en la GPU dos matrices inicializadas de forma aleatoria, pero en lugar de comparar la salida, se guarda en un fichero cuyo nombre es el indicado en el primer argumento.
 - c. Dos argumentos:
La aplicación inicializa las matrices con los valores que toma de los dos ficheros que se dan como argumentos. No se escribe nada en ningún fichero.
 - d. Tres argumentos:
La aplicación leerá los valores con los que inicializa las matrices de los ficheros cuyo nombre coincida con los dos primeros argumentos, y escribe el resultado en el fichero indicado por el tercer argumento.

Si deseáis utilizar la salida de una ejecución como la entrada de otra, es necesario eliminar la primera línea del fichero, en la que se indica la precisión de los valores de salida en ese fichero (por ahora ese valor no es importante para nosotros, pero haremos uso de él más adelante)

1.1 Cuestiones sobre la implementación de la multiplicación de matrices básica

En la memoria es necesario además contestar a las siguientes preguntas:

1. ¿Cuántas veces es cargado cada elemento de las matrices de entrada en el código ejecutado en la GPU (el kernel)?
2. ¿Cuál es el ratio entre accesos a memoria y cálculos en coma flotante en cada hebra? Considera la multiplicación y la suma como operaciones diferentes e ignora el almacenamiento del resultado en la cuenta de accesos a memoria. Supón que todas las variables locales están guardadas como si fueran registros.

1.2 Evaluación de esta parte

1. Funcionamiento y conocimiento (25%):
 - a. Devuelve el resultado correcto para las matrices de entrada dadas

2. Funcionalidad y elegancia (40%)
 - a. Uso correcto de las llamadas a las funciones de las librerías de CUDA y las extensiones de C
 - b. Uso correcto de los identificadores de hebras para la multiplicación de matrices
3. Memoria
 - a. Redacción de la memoria (8%)
 - b. Respuesta a la primera pregunta (11%)
 - c. Respuesta a la segunda pregunta (16%)

2. Notas sobre reducción de vectores

1. Descomprimir el fichero practica7_reduccion.rar en <Dir_proyecto>\NVIDIA CUDA SDK\projects (o lo más parecido posible -ya que en los ordenadores del laboratorio ese directorio está protegido- para evitar problemas con las dependencias de ficheros)
2. Edita los ficheros fuente vector_reduction.cu y vector_reduction_kernel.cu para completar la funcionalidad necesaria para llevar a cabo la suma y reducción paralela en el device. El tamaño del array será siempre de 512 elementos en esta práctica (para facilitar el planteamiento de la implementación).
3. Tenemos dos modos de operación en esta aplicación:

- a. Sin argumentos

La aplicación creará un array inicializado aleatoriamente a procesar. Después de que el kernel del device sea invocado, realiza el mismo cálculo en la CPU y compara ambas soluciones. Si coincide (dentro de una cierta tolerancia) imprimirá “Test PASSED” por pantalla antes de terminar.

- b. Con un argumento

La aplicación inicializará el array de entrada con los datos que encuentre en el fichero cuyo nombre se indica como argumento.

En cualquier caso el programa muestra por pantalla el resultado final de los cálculos realizados en la GPU y en la CPU, supere o no el test de comparación.

2.1 Cuestiones sobre la implementación de la reducción de vectores

En la memoria es necesario además contestar a las siguientes preguntas:

1. ¿Cuántas veces es el bloque de threads ha de sincronizarse para reducir el array de 512 elementos a un único valor?
2. A lo largo de la vida del programa ¿Cuántos waps se verán afectados negativamente debido a la divergencia SIMD? Recuerda que la divergencia SIMD ocurre cuando hay hebras dentro de un mismo warp que toman caminos distintos de código en el programa.
3. ¿Cuál es el ratio entre accesos a memoria y cálculos en coma flotante en cada hebra? Considera la multiplicación y la suma como operaciones diferentes e ignora el almacenamiento del resultado en la cuenta de accesos a memoria. Supón que todas las variables locales están guardadas como si fueran registros.

2.2 Evaluación de esta parte

1. Funcionamiento y conocimiento (25%):

- a. Devuelve el resultado correcto para los ficheros de entrada dados
2. Funcionalidad y elegancia (40%)
 - a. Uso correcto de las llamadas a las funciones de las librerías de CUDA y las extensiones de C
 - b. Uso correcto de los identificadores de hebras para la multiplicación de matrices
3. Memoria
 - a. Redacción de la memoria (8%)
 - b. Respuesta a la primera pregunta (11%)
 - c. Respuesta a la segunda pregunta (16%)

Forma y fecha de entrega

Se entregará la memoria en el formato de cualquier procesador de textos (ó PDF) debidamente identificada junto con los ficheros con los programas en Cg en formato texto, tal y como los utilizaríais en el entorno de trabajo que se ha presentado en esta práctica.

Puede enviarse directamente mediante la aplicación correspondiente en el Campus Virtual dentro del plazo, e indicando “[PG Practica07]”. No olvidéis indicar vuestros nombres y apellidos en el cuerpo del mensaje.

La fecha tope de entrega de esta práctica será el **23 de Abril** de 2008.
Planificaros bien.

Nota aclaratoria

Todas las marcas y productos mencionados en este enunciado de prácticas están registradas por sus respectivas compañías, y su uso es de carácter descriptivo con fines docentes.

La práctica se basa en uno de los Machine Problems de iniciación a CUDA ideado por David Kirk, John Stratton y Kuangwei Hwang.