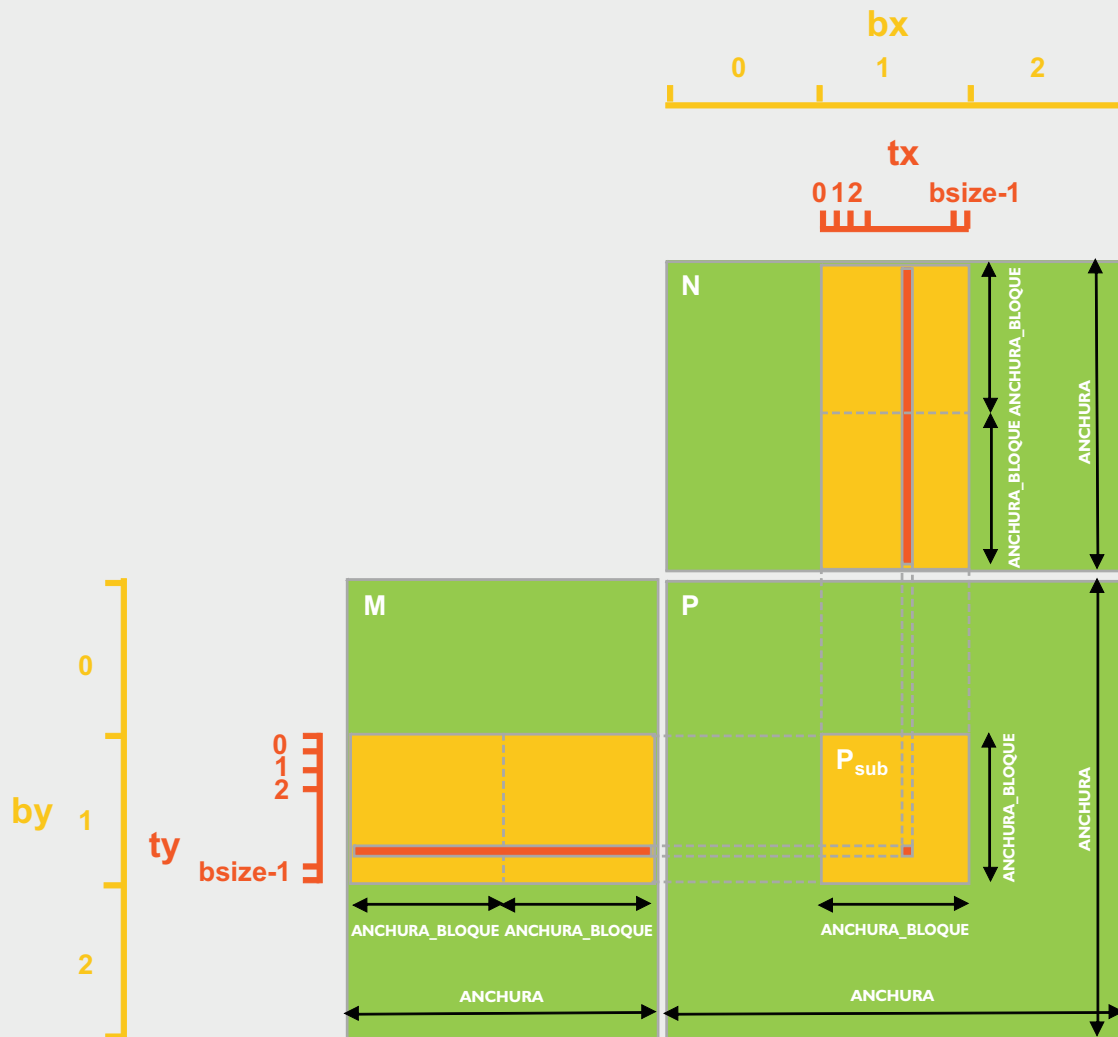


Práctica 8

Multiplicación de matrices eficiente en CUDA



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

**Procesadores
Gráficos**



Universidad
Rey Juan Carlos

Simplicity is the ultimate sophistication.
-- Leonardo da Vinci

“Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated.”
-- Edsger Dijkstra

“Simple things should be simple, complex things should be possible.”
-- Alan Kay

Introducción

El objetivo de esta práctica consiste en ir un poco más allá de la mera programación paralela y tratar de hacer nuestro código (multiplicación de matrices de la práctica anterior) escalable para un mayor número de procesadores y tan rápido como sea posible manteniendo la precisión en los resultados. Para ello será necesario dividir el problema en una paralelización por datos así como tener muy en cuenta la arquitectura a la que nos estamos enfrentando, así como la jerarquía de memoria de CUDA y su aprovechamiento en nuestro favor.

En este caso, es preciso leer cuidadosamente los capítulos 4 y 5 del manual de programación de CUDA, que podéis encontrar aquí:
http://www.nvidia.com/object/cuda_develop.html

El ejemplo de algoritmos que vamos a tratar en esta práctica muy sencillo y CUDA trae librerías que lo realizan de forma mucho más eficiente, pero hemos decidido mantenerlo dado que la multiplicación de matrices es muy socorrida en la enseñanza de algorítmica en sistemas distribuidos y también nos puede servir para comparar ciertos sistemas.

Existen estudios muy interesantes acerca de cómo implementar este algoritmo de la forma más eficiente posible, entre los que destaca el artículo de K. Fatahalian, J. Sugerman y P. Hanrahan titulado: “Understanding the efficiency of GPU algorithms for matrix-matrix multiplication”, que recomendamos encarecidamente leer.

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

2. Notas sobre multiplicación de matrices por trozos

1. Descomprimir el fichero `multiplicacion_matrices_a_trozos.rar` en `<Proj_Dir>\NVIDIA CUDA SDK\projects`
2. Edita el fichero fuente de `matrixmul.cu` and `matrixmul_kernel.cu` para completar la funcionalidad de la multiplicación de matrices en la GPU (device). Las matrices pueden ser de cualquier tamaño, pero al menos podeis estar seguro de que la matriz resultante no será mayor de 64000 elementos.
3. De nuevo, en esta aplicación tendremos diferentes comportamientos en función de los parámetros de entrada por consola (el fichero de interfaz ha sido modificado respecto a la práctica anterior para que se pueda leer el tamaño de la matriz)
 - a. Sin argumentos
La aplicación crea dos arrays de dimensiones aleatorias y las inicializa de modo que la operación $M*N$ sea válida. Se reserva espacio suficiente para poder albergar el resultado en P. Una vez la multiplicación de matrices se ha calculado en la GPU, se repiten los cálculos en la CPU y se comparan los resultados. Si están dentro de un pequeño margen imprimirá el mensaje "Test PASSED" antes de terminar el programa.
 - b. Un argumento
La aplicación realiza la inicialización de forma completamente aleatoria, como en el caso anterior, pero se guarda el resultado en un fichero cuyo nombre vendrá indicado por el argumento.
 - c. Tres argumentos
La aplicación leerá las matrices de entrada de los ficheros indicados por los argumentos.
El primer argumento ha de contener tres enteros que serán utilizados como: `M.height`, `M.width`, and `N.height`
El segundo y tercer argumentos serán los ficheros con las entradas justas para rellenar las matrices M y N. No se escribe ninguna salida a fichero.
 - d. Cuatro argumentos
Idéntico al anterior, pero con un argumento extra para indicar en qué fichero se ha de guardar el resultado de la operación.

De nuevo para poder utilizar como entrada el resultado de una iteración anterior, es necesario borrar la primera línea del fichero de salida que muestra la precisión de los resultados escritos (este vaor no es relevante para la aplicación relativa a esta práctica)

A la hora de hacer los cálculos de cuántas hebras se pueden estar ejecutando simultáneamente en la GeForce 8800GTX de referencia, ten en cuenta que cada multiprocesador tiene 16 procesadores escalares. Supón también de cara a la implementación que cada variable local ocupa un registro de 32 bits y que no hay más registros reservados por el compilador.

2.1 Cuestiones sobre la implementación de la multiplicación de matrices

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

1. Explica con detalle la aproximación tomada, las particiones de los datos, etc...

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. Funciona correctamente en una GeForce 8800 GTX
 - b. Utilización óptima de la memoria compartida en el kernel para “esconder” el efecto de la latencia
3. Memoria
 - a. Redacción de la memoria en la que se cuenten las dificultades y decisiones tomadas (15%)
4. **Hall of Fame** (+10%)
 - a. Será tenido en cuenta el tiempo de ejecución del algoritmo en comparación con el de vuestros compañeros. Esta parte puede contar más de un 10% si se destaca notablemente.
Los ganadores, cuyo código sea más eficiente, se pondrán en la página web de la asignatura.

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 **30 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.