TRABAJO PRÁCTICO

DE

TAI II

TEMA: NVIDIA CUDA

PROFESOR: JUAN DE URRAZA

PABLO DANIEL MARIN A

MAT: 45520

AÑO 2008

INDICE

INTRODUCCION

CUDA en cualquier ordenador puede convertirlo en un superordenador ya que potenciaría de forma importante los numerosos procesadores que están alojados en el núcleo de su tarjeta gráfica. El hecho de que los datos de proceso se realicen en paralelo hace que la velocidad de ejecución sea 400 veces superior a la de un ordenador sin este software.

CUDA tiene el potencial de ser una buena herramienta en las industrias de GPU y CPU y aplicaciones que pueden darle al hardware de NVIDIA una ventaja sobre otras GPU's y resalta la necesidad de que los consumidores optimicen sus PCs para que tengan un poder decente de la CPU y la GPU.

CUDA es el único entorno basado en el lenguaje C que permite a los programadores escribir software para resolver problemas computacionales complejos en menos tiempo, aprovechando la gran capacidad de procesamiento paralelo de las GPU multinúcleo, es decir, que aprovecha la gran capacidad de procesamiento de las GPU para resolver los problemas de cálculo más complejos y de mayor carga computacional.

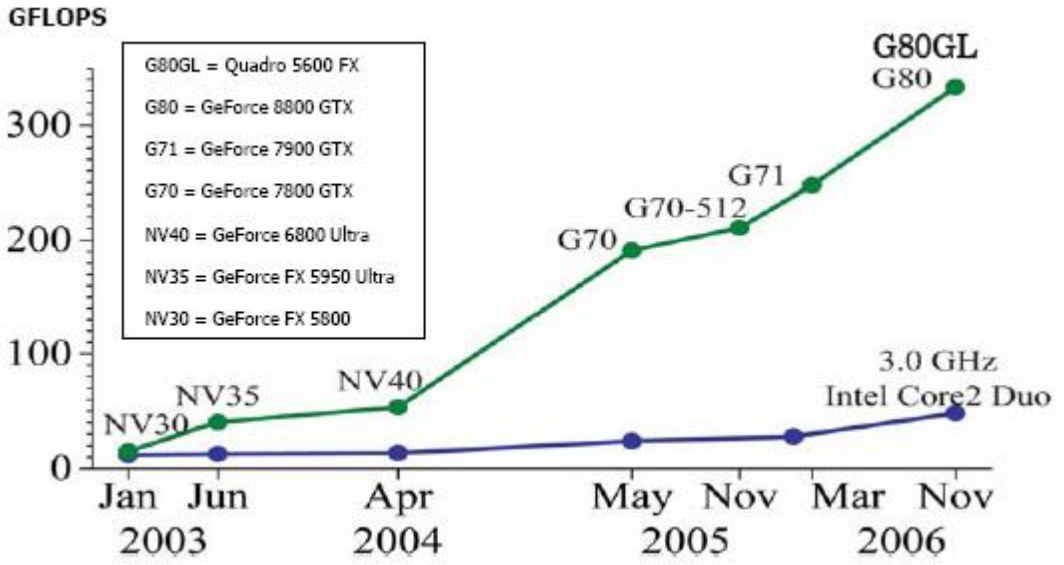
Miles de programadores están utilizando ya las herramientas de desarrollo de CUDA a fin de acelerar todo tipo de aplicaciones, desde herramientas de codificación de audio y vídeo, hasta software para la exploración de gas y petróleo, el diseño de productos, la generación de imágenes en medicina o la investigación científica.

GPU

LA UNIDAD DE PROCESAMIENTO GRAFICO COMO UN DISPOSITIVO DE PROCESAMIENTO DE DATOS EN PARALELO

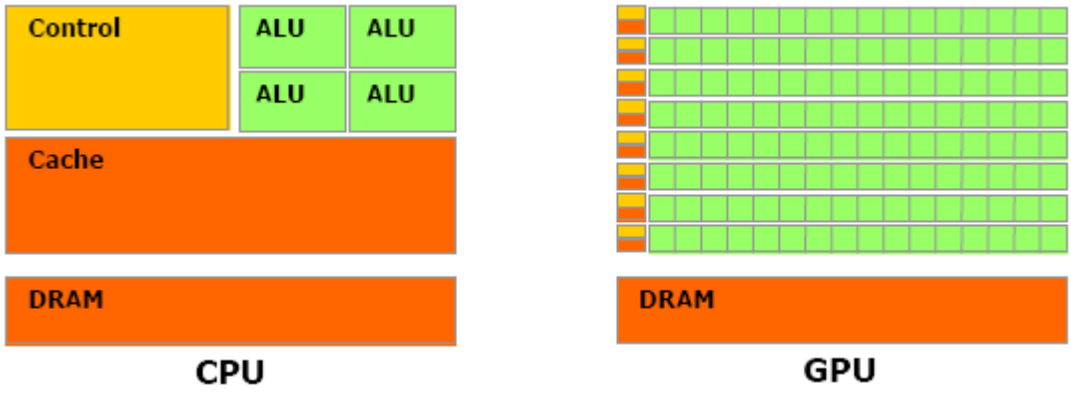
En solo unos pocos años, la Unidad de Procesamiento Gráficos Programable ha evolucionado a un potente procesador de cómputos, con múltiples núcleos manejados por memorias con mucho ancho de banda. Las GPUs de la actualidad ofrecen una increíble solución para procesamiento de gráficos y no gráficos en general.

El siguiente grafico muestra la cantidad de operaciones de punto-flotante por segundo de una CPU y una GPU.



La razón principal de esta rápida evolución es que la GPU es especial para el computo intensivo, rápido y en paralelo, que es exactamente lo que se necesita para

el procesamiento de gráficos, por lo cual está diseñado con muchos más transistores destinados al procesamiento de datos, en vez de almacenamiento en cache de datos, control de flujos, etc, a diferencia de la CPU.



La GPU es adecuado para abordar los problemas que puede expresarse como cómputos de datos en paralelo (el mismo programa se ejecuta en muchos de los elementos de datos en forma paralela) con mayor intensidad de las operaciones aritméticas sobre las operaciones de memoria. Debido a que el mismo programa se ejecuta para cada elemento de dato, hay una menor necesidad de control de flujos sofisticados, y ya que es ejecutado en muchos de los elementos de datos y tiene una mayor intensidad de la aritmética, la latencia de acceso a memoria es sustituida con cálculos, en lugar de grandes caches destinados a datos.

El procesamiento de datos en paralelo mapea los elementos de datos a procesamiento de hilos en paralelos.

Muchas aplicaciones que procesen grandes conjuntos de datos, tales como vectores (arrays) pueden utilizar un modelo de programación paralela para acelerar los cálculos.

En 3D, grandes conjuntos de píxeles y vértices se asignan a los hilos en paralelo. Del mismo modo, el post-procesado de imágenes, vídeo de codificación y decodificación, la imagen de escala, visión estéreo, y el reconocimiento de patrones pueden mapear bloques de imágenes y bloques de píxeles a procesamiento de hilos en paralelos. De hecho, muchos algoritmos fuera del campo de la imagen y la prestación de tratamiento son acelerados por procesamiento de datos en paralelo, como el procesamiento de señales o la física de simulación para las finanzas computacionales o la biología computacional.

PROBLEMAS

Hasta ahora, el acceso a todas esas ventajas computacionales del la GPU y el aprovechamiento eficiente para aplicaciones no gráficas sigue siendo difícil de implementar debido a que:

* La GPU sólo puede ser programado a través de un API de gráficos, la imposición de una elevada curva de aprendizaje a los novatos y la sobrecarga de un API insuficiente a las aplicaciones no gráficas.

* La GPU DRAM puede ser leído de una manera general, programas GPU pueden reunir los elementos de datos desde cualquier parte de DRAM, pero no pueden ser escritos de una manera general a cualquier parte de DRAM, esto elimina una gran cantidad de la flexibilidad de programación disponible en la CPU.

CUDA es una respuesta directa a estos problemas y expone a la GPU como un verdadero dispositivo de cómputo de datos en paralelo.

NUEVA ARQUITECTURA DE COMPUTOS EN LA GPU

CUDA es una nueva arquitectura de hardware y software para gestión de los cálculos sobre la GPU como dispositivo de procesamiento paralelo sin la necesidad de mapearlos un API de gráficos.

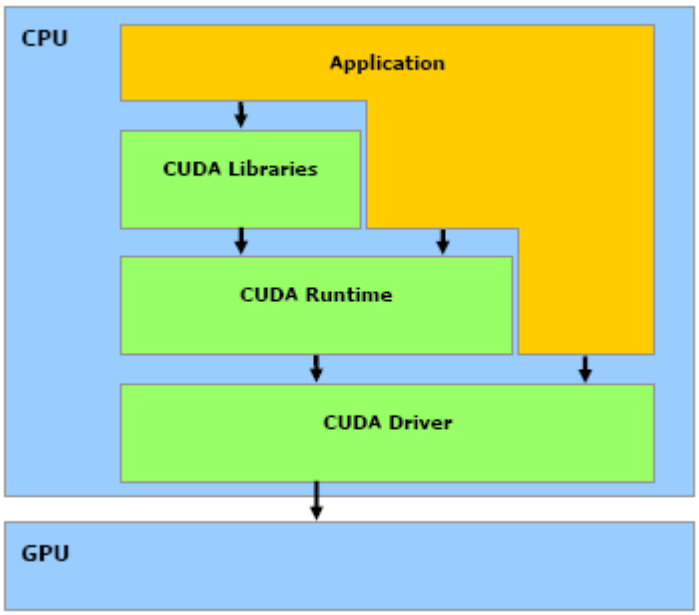
Está disponible para las GeForce 8 Series, Quadro FX 5600/4600, y soluciones Tesla. El sistema operativo multitarea del dispositivo es responsable de la gestión del acceso a la GPU por varios CUDA y aplicaciones gráficas que se ejecutan simultáneamente.

La pila de software se compone de las siguientes capas:

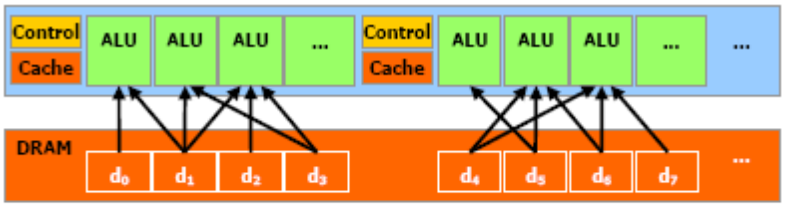
*Un controlador de hardware

*Una interfaz de programación de aplicaciones (API) y su tiempo de ejecución

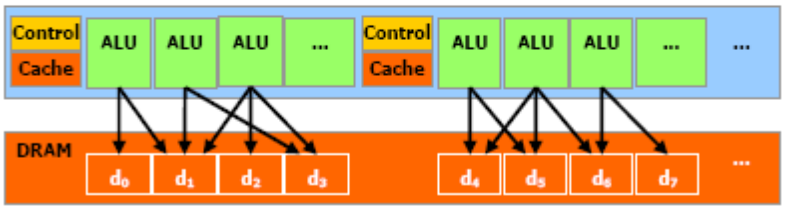
*Dos librerías matemáticas de uso común, CUFFT y CUBLAS



CUDA provee un direccionamiento de DRAM (mostrado abajo en la grafica) para mayor flexibilidad en la programación, desde el punto de vista de lectura y escritura en cualquier dirección de la DRAM, igual que en un CPU.

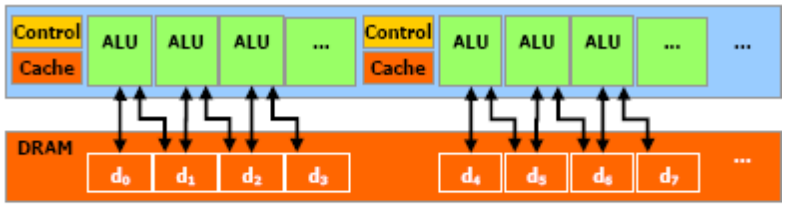


Gather

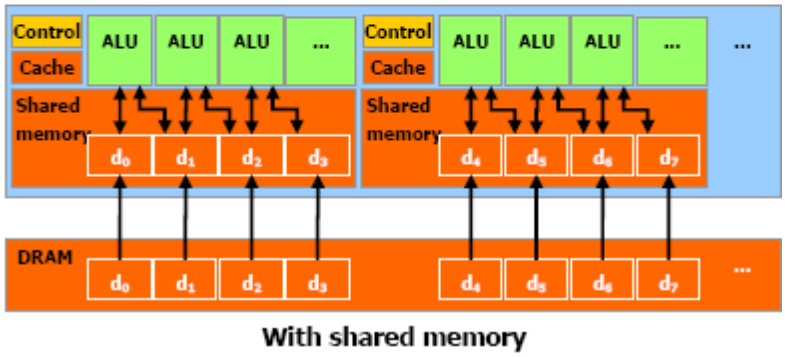


Scatter

CUDA tiene una memoria de acceso muy rápido de lectura y escritura on-chip, que es utilizada por los hilos para intercambiar datos entre ellos.



Without shared memory



With shared memory

MODELO DE PROGRAMACION

Un coprocesador de múltiples hilos

Cuando se programa con CUDA, la GPU se considera como un dispositivo de cómputo capaz de ejecutar un número muy elevado de los hilos en paralelo.

Funciona como un coprocesador de la CPU, o del host, en otras palabras, una parte de una aplicación que se ejecuta muchas veces, pero en datos independientes diferentes, puede ser aislado en una función que se ejecuta en el dispositivo como muchos hilos diferentes. Para eso, esta función se compila con el conjunto de instrucciones del dispositivo y el programa resultante, kernel, se descarga al dispositivo.

Tanto el host y el dispositivo mantienen sus propias DRAM, la memoria del host y la memoria del dispositivo, respectivamente. Uno puede copiar datos desde una DRAM a los demás a través de llamadas a la API que utilizan *Acceso Directo a Memoria (DMA).*

Lote de hilos

El lote de hilos (treads) que ejecuta el kernel está organizado como una red (grid) de bloques (blocks) de hilos.

Bloque de hilos

Un bloque es un lote de hilos que pueden interactuar juntos de manera eficiente intercambiando datos rápidamente a través de memorias compartidas y sincronizar su ejecución a fin de coordinar accesos a memoria. Uno puede especificar los puntos de sincronización en el *kernel*, donde los bloques están suspendidos hasta que todos llegan al punto de sincronización.

Cada hilo se identifica por su *ID de hilo,* que es el número de hilo dentro del bloque. Para ayudar los complejos ordenamientos de *los ID de hilo,* una aplicación puede especificar un bloque de dos o tres dimensiones de tamaño arbitrario e identificar cada uno utilizando un índice relacionado con su lugar.

Para un bloque bidimensional de tamaño (Dx, Dy), el *ID del hilo* es (xy Dx) y para uno de tres dimensiones de tamaño (DX, DY, DZ), el *ID de hilo* es (Dx xy z Dx Dy).

Red de bloques de hilos

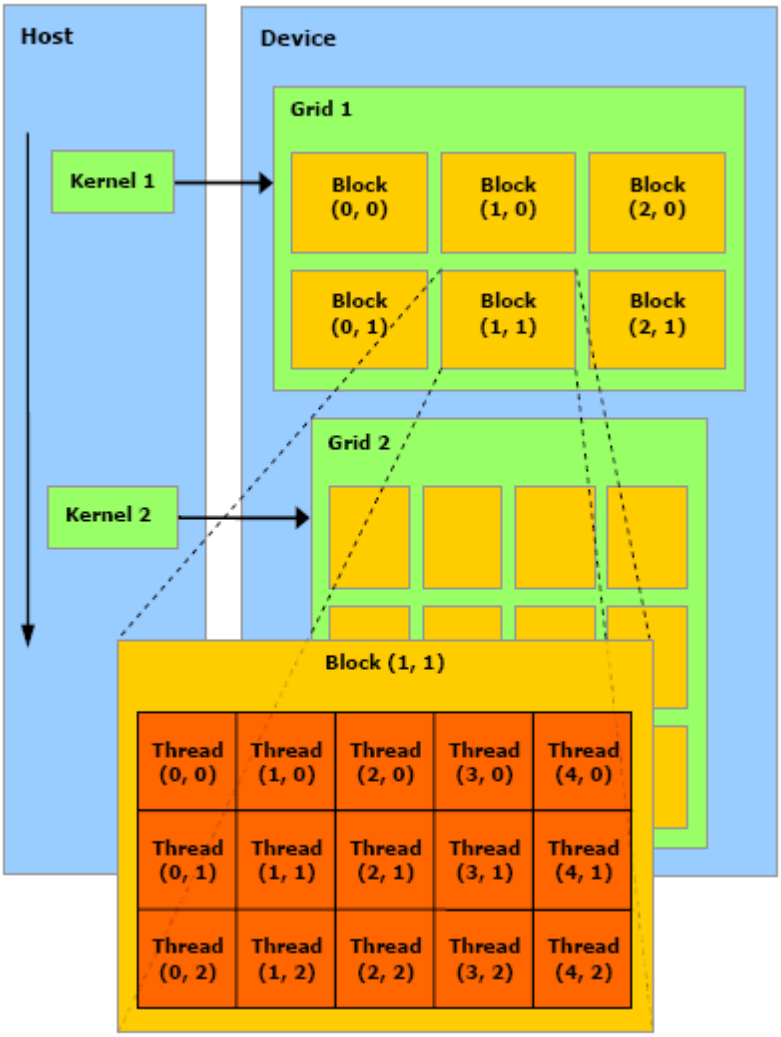
Hay un número máximo limitado de hilos que un bloque puede contener. Sin embargo, los bloques del mismo tamaño y dimensión, que ejecutan el mismo kernel pueden agruparse en una red de bloques, a fin de que el número total de los hilos que pueden ser corridos en una sola invocación del kernel sea mucho mayor.

Los hilos en diferentes bloques de la misma red no pueden comunicarse y sincronizar unos con otros. Este modelo permite que el kernel se ejecute de manera

eficiente sin y re compilación en diversos dispositivos con diferentes capacidades paralelas:

Un dispositivo puede ejecutar todos los bloques de una red en forma secuencial si tiene muy poca capacidad de procesamiento paralelo, y en paralelo si tiene mucha capacidad, o por lo general una combinación de ambos.

Cada bloque se identifica por su *ID de bloque*, que es el número de bloque dentro de la red. Para ayudar al ordenamiento, una aplicación puede especificar una red como una matriz bidimensional de tamaño arbitrario e identificar cada bloque utilizando un índice relacionado con su ubicación como en el caso de los bloques.

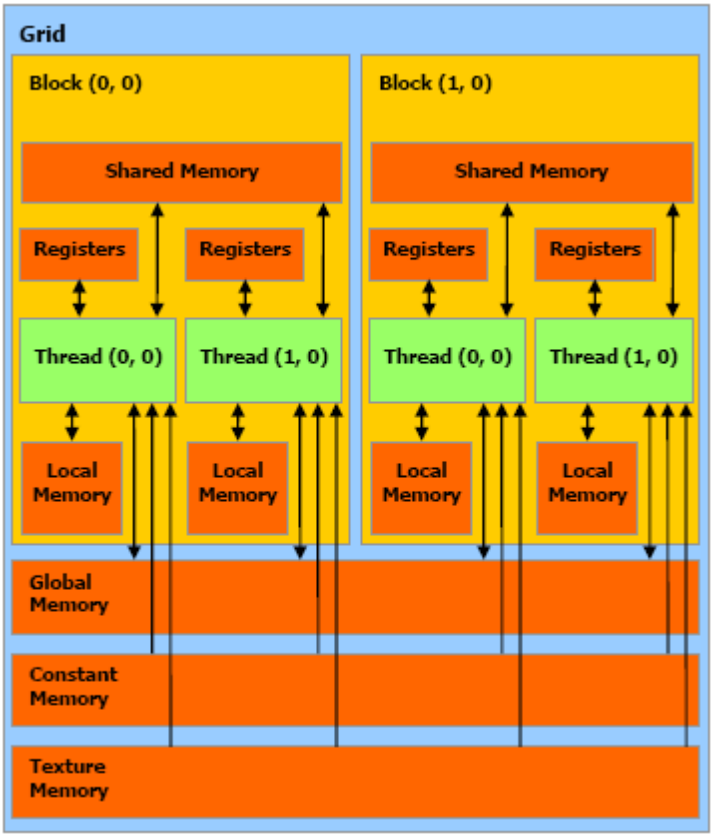

Host
Device
Grid 1
Kernel 1
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 2
Kernel 2
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)

MODELO DE MEMORIA

Un hilo que está siendo ejecutado por un dispositivo solo tiene acceso a la DRAM de ese dispositivo o a la memoria on-chip a través de los siguientes espacios de memoria

- Lectura-Escritura por hilo de registros
- Lectura-Escritura por hilo de memoria local
- Lectura-Escritura por bloques de memoria compartida
- Lectura-Escritura por conjunto de memoria global
- Lectura por conjunto de memorias constantes
- Lectura por conjunto de memorias de textura

Las memorias constantes, globales y textura, pueden ser leídas y escritas por el host y permanecen el kernel de la misma aplicación.


Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Shared Memory
Registers
Registers
Registers
Registers
Thread (0, 0)
Thread (1, 0)
Thread (0, 0)
Thread (1, 0)
Local Memory
Local Memory
Local Memory
Local Memory
Global Memory
Constant Memory
Texture Memory

DESDE EL PUNTO DE VISTA DEL HARDWARE

Un conjunto de Multiprocesadores de memoria compartida

Cada multiprocesador tiene una sola instrucción, de datos de arquitecturas múltiples (SIMD): En cualquier ciclo de reloj, cada procesador del multiprocesador ejecuta la misma instrucción, pero opera en diferentes datos.

Cada multiprocesador de memoria on-chip es uno de los siguientes cuatro tipos:

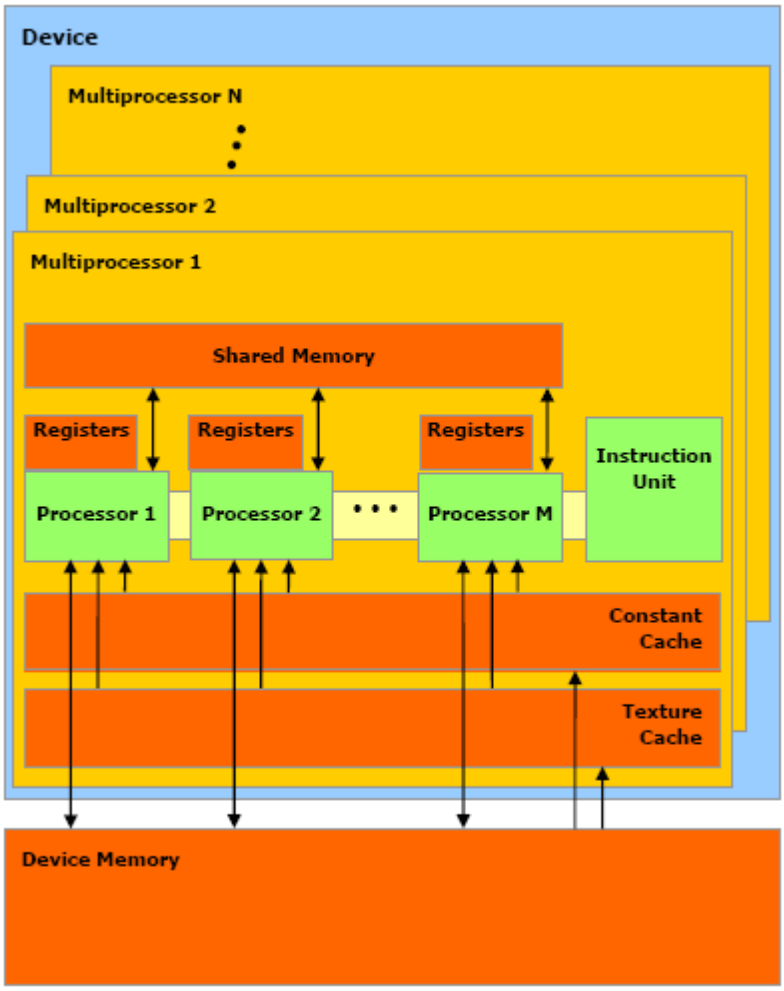
* Un conjunto de registros locales de 32-bit por procesador,

* Un *cache de datos en paralelo* o de *memoria compartida* que es compartida por todos los procesadores e implementan el *espacio de memoria compartida*,

* Una *texture cache* de sólo lectura que es compartida por todos los procesadores y que se implementa como una región de solo lectura en la memoria del dispositivo,

* Una *cache constante* de sólo lectura que es compartida por todos los procesadores y que se implementa como una región de solo lectura en la memoria del dispositivo,

Los espacios de memorias local y global se implementan como lectura-escritura de regiones de la memoria del dispositivo y no están en caché. El dispositivo se implementa como un conjunto de multiprocesadores como se ilustra en la Figura más abajo


Device
Multiprocessor N
Multiprocessor 2
Multiprocessor 1
Shared Memory
Registers
Registers
Registers
Instruction Unit
Processor 1
Processor 2
Processor M
Constant Cache
Texture Cache
Device Memory

API

Una extensión del lenguaje de programación C

El objetivo de la interfaz de programación de CUDA es proporcionar un camino relativamente sencillo para los usuarios familiarizados con el lenguaje de programación C para escribir fácilmente programas para su ejecución por el dispositivo.

Se compone de:

* Un conjunto mínimo de extensiones para el lenguaje C, que permiten que el programador ejecute porciones del código fuente para el dispositivo.

* Una biblioteca en tiempo de ejecución que se dividió en:

    * Un componente de host, que se ejecuta en el host y proporciona funciones de control de acceso y uno o más dispositivos

    * Un componente del dispositivo, que se ejecuta en el dispositivo y proporciona funciones específicas del dispositivo.

* Un componente común, que incorpora en el vector de tipos y un subconjunto de la biblioteca C estándar que es soportado por el host y el dispositivo.

Cabe destacar que solos las funciones de la biblioteca C estándar que son soportadas por el dispositivo son las funciones proporcionadas por el componente común en tiempo de ejecución.

CARACTERISTICAS DE LA TECNOLOGIA CUDA

* Lenguaje C estándar para el desarrollo de aplicaciones de procesamiento paralelo en la GPU.

* Librerías numéricas estándar para FFT (Fast Fourier Transform) y BLAS (Basic Linear Algebra Subroutines).

* Controlador CUDA dedicado a cálculo con comunicación de datos de alta velocidad entre la GPU y la CPU.

* El controlador de CUDA interacciona con los controladores de gráficos OpenGL y DirectX.

* Compatibilidad con sistemas operativos Linux de 32/64 bits y Windows XP de 32/64 bits.

CUDA está disponible para Linux, Mac OS X y Windows, y requiere una placa de video NVidia de la línea **GeForce** (8400M a 9800 GX2), **Quadro** (NVS 130M a FX 5600) o Tesla (S870, D870 o C870).

SDK PARA PROGRAMACION EN CUDA

El kit de desarrollo (SDK) proporciona ejemplos con código fuente para ayudar a los programadores a familiarizarse con CUDA.

Ejemplos:

- Ordenación bitónica en paralelo

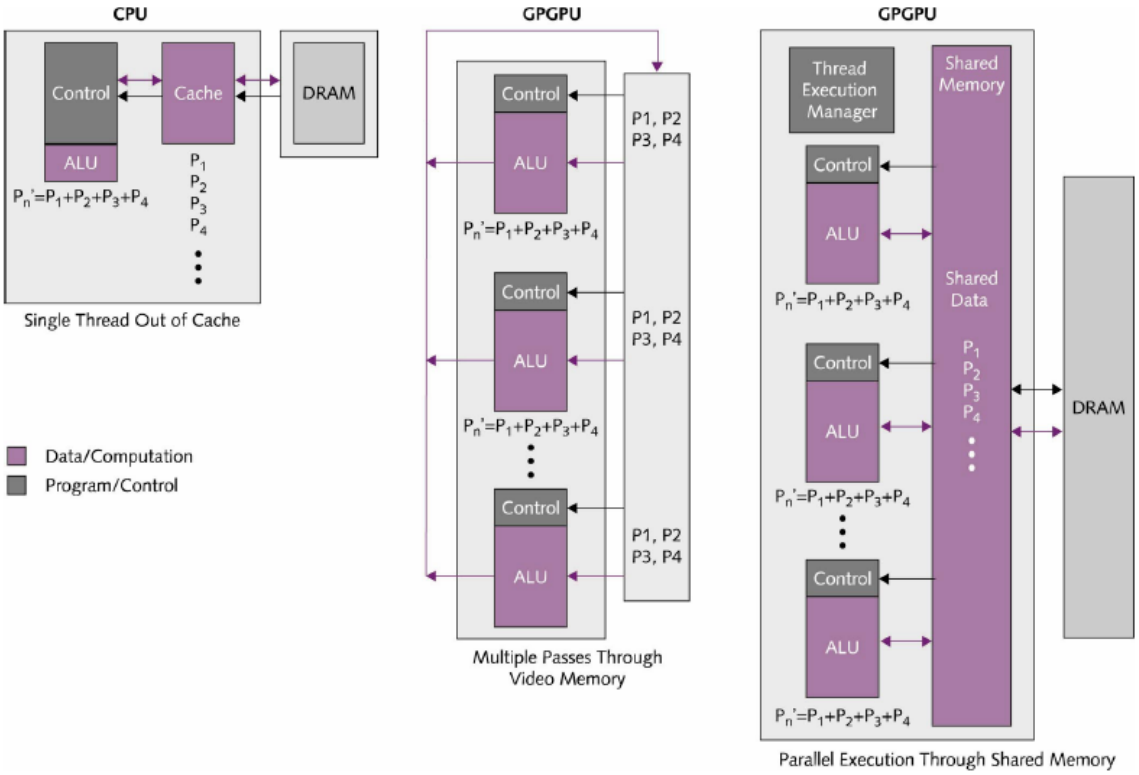
- Multiplicación de matrices
- Trasposición de matrices
- Análisis del rendimiento mediante temporizadores
- Suma de prefijos de arrays grandes (modelo Scan) en paralelo
- Convolución de imágenes
- DWT 1D mediante la transformada wavelet de Haar
- Ejemplos de interacción de gráficos OpenGL y Direct3D
- Ejemplos de uso de las librerías BLAS y FFT en CUDA
- Integración de código C y C++ para CPU-GPU
- Modelo binomial de valoración de opciones (BOPM)
- Valoración de opciones con el modelo de Black-Scholes
- Valoración de opciones mediante el método Montecarlo
- Mersenne Twister paralelo (generación de números aleatorios)
- Cálculo de histogramas en paralelo
- Supresión de ruido en las imágenes
- Filtro Sobel para detección de bordes
- Plug-in MATLAB® de MathWorks

EL MODELO DE PROGRAMACION EN CUDA

Es diferente al código de CPU y a los códigos de procesamiento paralelo actual que fueron utilizados para programar la CPU.

Los diferentes tipos de procesamientos

- CPU: ejecuta un solo hilo de ejecución recogiendo una sola instrucción de la cache
- GPGPU: utiliza la GPU para procesamiento de propósito general, es altamente paralelo pero tiene muchos accesos a memoria.
- CUDA comparte los datos a través de una memoria compartida


CPU
Control
Cache
DRAM
ALU
$P_n'=P_1+P_2+P_3+P_4$
P_1 P_2 P_3 P_4
Single Thread Out of Cache
Data/Computation
Program/Control
GPGPU
Control
ALU
P1, P2
P3, P4
$P_n'=P_1+P_2+P_3+P_4$
Multiple Passes Through
Video Memory
GPGPU
Thread
Execution
Manager
Shared
Memory
Shared
Data
DRAM
Parallel Execution Through Shared Memory

BIBLIOGRAFIA

PARALLEL PROCESSINGWITH CUDA

CUDA Programming Guide Version 1.0

www.nvidia.com

www.tomshardware.com

www.gpgpu.org

www.chw.net

www.laflecha.net

www.easyboinc.org/