TALLER DE PROGRAMACIÓN SOBRE GPUS

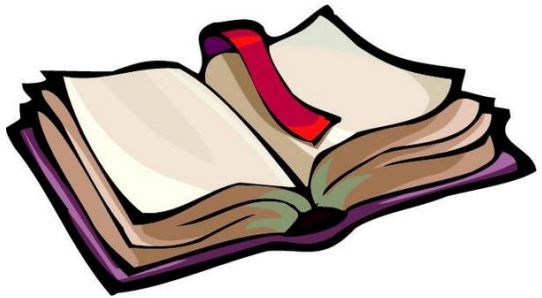
Facultad de Informática – Universidad Nacional de La Plata

Dr. Adrián Pousa

Capability - Jerarquia de memoria Nvidia

Agenda

I. *CUDA Capability*
II. *Características generales de la jerarquía de memoria Nvidia*
III. *Descripción de las distintas memorias de la jerarquía NVidia*
    I. *Memoria global*
    II. *Memoria de constantes*
    III. *Memoria de texturas*
    IV. *Memoria compartida*
    V. *Registros*
    VI. *Memoria local*
IV. *Resumen*
V. *Limitaciones*



Agenda

I. **CUDA Capability**

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   I. Memoria global

   II. Memoria de constantes

   III. Memoria de texturas

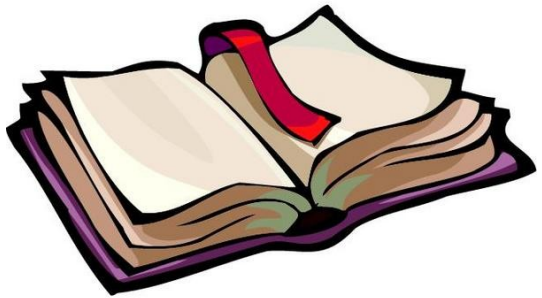
   IV. Memoria compartida

   V. Registros

   VI. Memoria local

IV. Resumen

V. Limitaciones



CUDA – Capability

- Una nueva capability indica una nueva generación de dispositivos, agrega funcionalidad y tiene compatibilidad hacia atrás.
- Capabilities existentes: 1.0, 1.1, 1.2, 1.3, 2.0, 2.1, 3.0, 3.5, 3.7, 5.0, 5.2, 5.3, 6.0, 6.1, 6.2, 7.0, 7.1 y 7.2.
- Una capability se representa por dos dígitos, el primero es la revisión y el segundo es una versión.
- El mismo número de revisión indica la misma arquitectura.

CUDA – Capability

- **Capability 1.0 (G80):** Inicio.
- **Capability 1.1:** soporte para funciones atómicas sobre memoria global y para palabras de 32 bits.
- **Capability 1.2:** permite funciones atómicas sobre memoria compartida, funciones atómicas sobre memoria global para palabras de 64 bits.
- **Capability 1.3:** permite operaciones sobre números de punto flotante de doble precisión.

CUDA – Capability

- **Capability 2.0** (Fermi):
  - Soporte para grid de tres dimensiones, operaciones de suma atómica de números de punto flotante sobre la memoria compartida y global para palabras de 32 bits
  - Distintas alternativas de sincronización de barreras (todas ellas implican la evaluación de condiciones)
  - Funciones de superficie

CUDA – Capability

- **Capability 3.0** (Kepler):
  - No afecta características CUDA, sólo cuan rápido corren los programas dado que las tarjetas tienen más cores

- **Capability 3.5, 3.7 (Big Kepler):**
  - Paralelismo dinámico
  - Incrementa la cantidad de registros por hilo a 255

CUDA – Capability

- **Capability 5.0, 5.2, 5.3** (Maxwell):
  - Modificaciones con respecto al ahorro de energía
  - Menor superficie
  - Reducción de componentes (registros, memoria compartida)
  - Mayor rendimiento con mejor eficiencia energética
- **Capability 6.0, 6.1, 6.2** (Pascal)
- **Capability 7.0, 7.1, 7.2** (Volta)
- **Mas información:**

  http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities

CUDA – Capability

- Las capabilities indican límites de memoria, hilos, bloques etc.

	Capacidad de Cómputo									
Especificaciones Técnicas	1.0	1.1	1.2	1.3	2.x	3.0	3.5	3.7	5.0	5.2
Máxima dimensión de un grid.	2				3					
Máxima dimensión x de un grid.	65535					$2^{31} - 1$				
Máxima dimensión y o z de un grid.	65535									
Máxima dimensión de un bloque.	3									
Máxima dimensión x o y de un bloque.	512				1024					
Máxima dimensión z de un bloque.	64									
Máximo número de threads por bloques.	512				1024					

CUDA – Capability en la sala

- **Nvidia geforce 560TI:**
  - Arquitectura Fermi GF114
  - Capability 2.1

- **Nvidia geforce 960:**
  - Arquitectura Maxwell GM206-300
  - Capability 5.2

- Descargar el archivo *gpu_info.cu* que utiliza la estructura *struct cudaDeviceProp* para acceder por código a las capabilities del dispositivo.

Agenda

I. *CUDA Capability*

II. ***Características generales de la jerarquía de memoria Nvidia***

III. *Descripción de las distintas memorias de la jerarquía NVidia*

   I. *Memoria global*

   II. *Memoria de constantes*

   III. *Memoria de texturas*

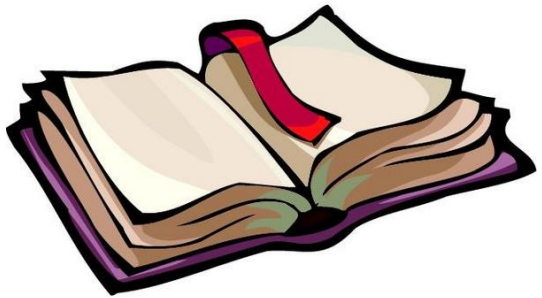
   IV. *Memoria compartida*

   V. *Registros*

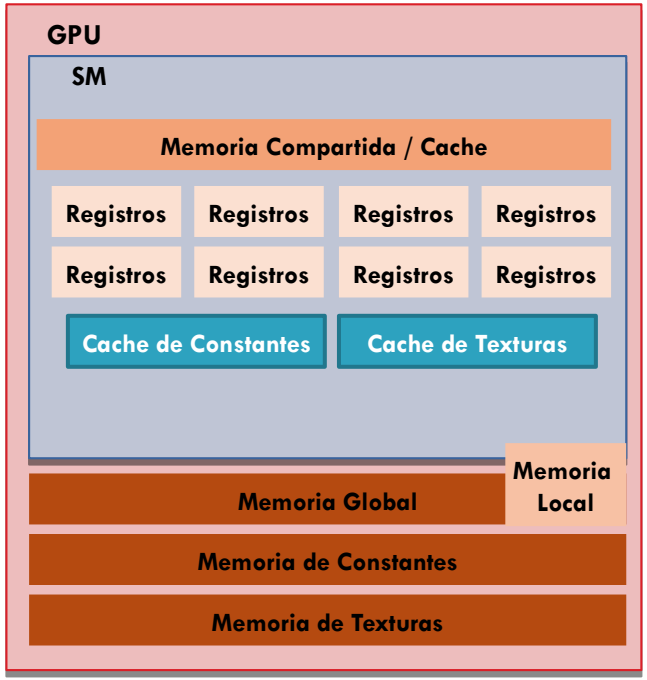
   VI. *Memoria local*

IV. *Resumen*

V. *Limitaciones*



Jerarquía de Memoria



Los hilos pueden acceder a datos que están en memorias:

- **Off-Chip:** Fuera de los SMs
  - Memoria global
  - Memoria de constantes
  - Memoria de texturas
  - Memoria local
- **On-Chip:** Dentro de cada SM
  - Registros
  - Memoria compartida (shared)
  - Distintas caché

Jerarquía de Memoria

- Desde el punto de vista de la arquitectura, cada nivel de memoria se diferencia en:

  - Tamaño (GB a bits).
  - Ancho de banda (TB a Mb).
  - Velocidad de acceso (Off-chip vs. On-chip):

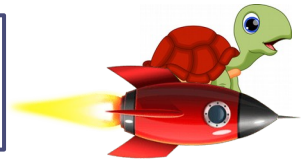
Jerarquía de Memoria

- De acuerdo a la velocidad de acceso, la jerarquía se ordena desde la memoria mas lenta a la mas rápida como:

Memoria Global
Memoria Local

Memoria de Constantes
Memoria de Texturas

Memoria Compartida
Registros



Jerarquía de Memoria

- Desde el punto de vista del programador, éste decide las características de las variables y en qué memoria las aloja.
- Esto determina:
  - La velocidad de acceso
  - La visibilidad (CPU y/o GPU)
  - **El alcance**
  - **El tiempo de vida**

Jerarquía de Memoria

- El **alcance** establece qué threads acceden a una variable:
  - Un único thread
  - Todos los threads de un bloque
  - Todos los threads de un grid
  - Todos los threads de una aplicación

Jerarquía de Memoria

- El **tiempo de vida** indica la duración de la variable en la ejecución del programa:

  - **La ejecución de un kernel:** la variable se crea durante la ejecución de un kernel y se destruye cuando el kernel finaliza
  - **Toda la aplicación:** Permanece y es accesible por todos los kernels involucrados en la aplicación

Jerarquía de Memoria

- La latencia de una instrucción de memoria varía dependiendo de:
  - El espacio de memoria al que se accede
  - El patrón de acceso

- Los cambios de contexto en la ejecución de un *thread* se producen cuando se ejecuta una instrucción de memoria, evitando así que el SM esté ocioso (**Multithreading por hardware**).

Agenda

I. *CUDA Capability*

II. *Características generales de la jerarquía de memoria Nvidia*

III. ***Descripción de las distintas memorias de la jerarquía NVidia***

   I. ***Memoria global***

   II. ***Memoria de constantes***

   III. ***Memoria de texturas***

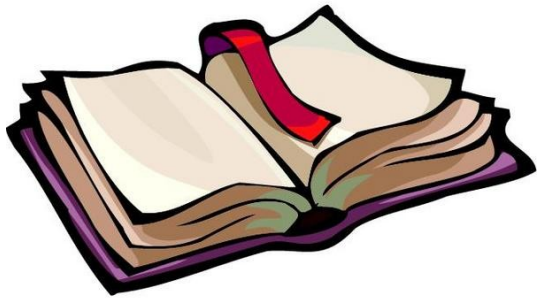
   IV. ***Memoria compartida***

   V. ***Registros***

   VI. ***Memoria local***

IV. *Resumen*

V. *Limitaciones*



Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. **Descripción de las distintas memorias de la jerarquía NVidia**

   I. **Memoria global**

   II. Memoria de constantes

   III. Memoria de texturas

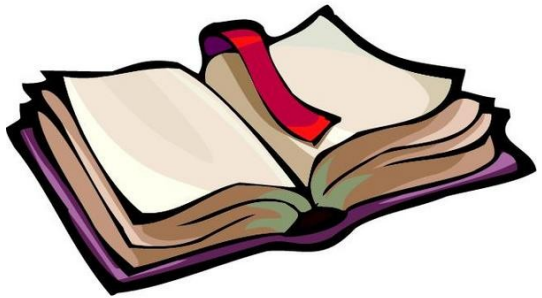
   IV. Memoria compartida

   V. Registros

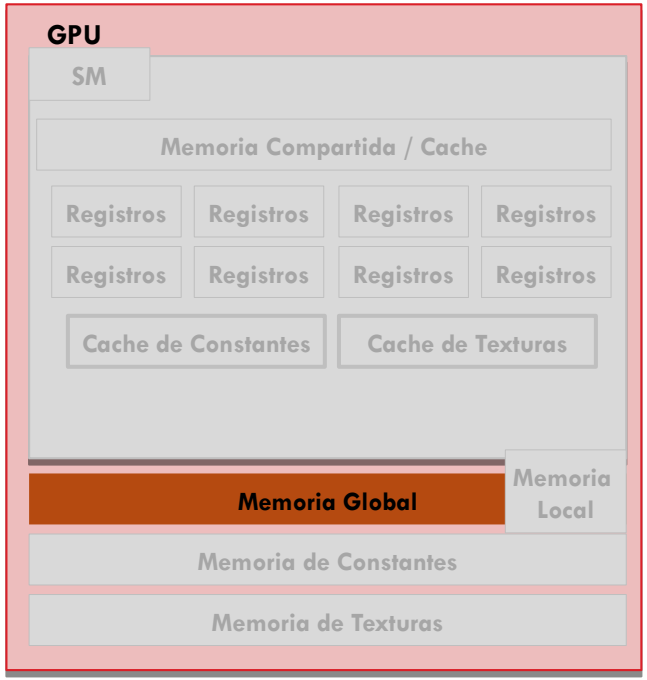
   VI. Memoria local

IV. Resumen

V. Limitaciones



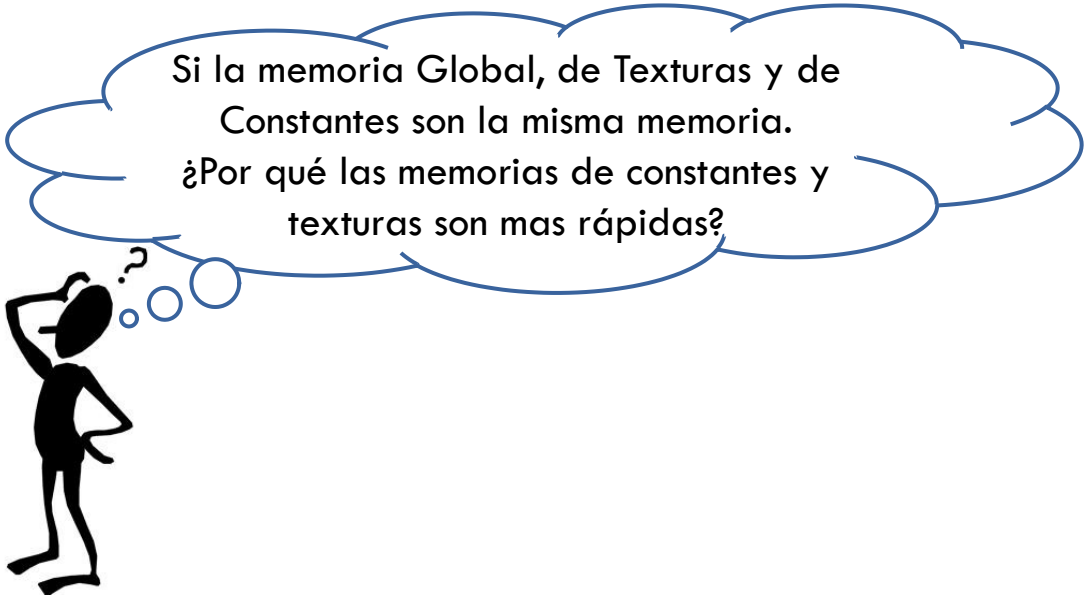
Memoria Global



- **Visibilidad:** lectura y escritura tanto por CPU como GPU.
- **Velocidad:** Es una memoria off-chip, por lo tanto su acceso es lento.
- **Variables:**
  - **Alcance:** *todos los threads de un grid y/o de una aplicación.*
  - **Tiempo de vida:** toda la aplicación.

Memoria Global

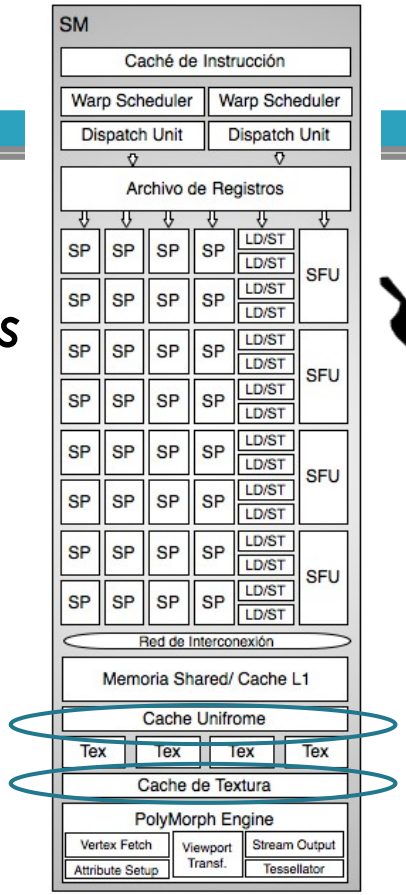
- *Se divide en tres espacios de memoria separada:*
  - Memoria Global.
  - Memoria de Textura.
  - Memoria de Constantes.



Memoria Global

Tanto la memoria de textura como la memoria de constantes poseen caché on-chip.

Además de ser de sólo lectura por los hilos de GPU.



Memoria Global

- Todos los threads pueden acceder a una variable global (declarada como __device__), puede verse como una variable compartida por todos los threads del grid.

- Las variables globales pueden utilizarse como medio de comunicación entre threads de distintos bloques. (no habitual por bajo rendimiento).

- El tiempo de vida de una variable global es toda la aplicación, una vez finalizado un kernel los valores de las variables globales modificadas persisten y pueden ser accedidos por los siguientes kernels.

Memoria Global

- CUDA tiene limitaciones respecto al uso de punteros a la memoria global.
- Los punteros se pueden usar de dos maneras:
  - Cuando un objeto es alojado en la memoria del dispositivo desde el Host con la función cudaMalloc() y pasado como parámetro en la invocación del kernel.
  - Asignarle la dirección de una variable global para trabajar sobre ella.
- No se pueden alocar punteros dentro del kernel.

Memoria Global

- Si bien el acceso es lento, la velocidad depende del patrón de acceso.

- Los accesos a la memoria global se resuelven en medio *warp*:
  - Cuando un warp intenta acceder a memoria global, se realiza un requerimiento para la primera mitad del *warp* y otro para la segunda mitad.

Memoria Global

- Para aumentar la eficiencia, el hardware puede **unificar las transacciones** dependiendo del patrón de acceso.

- En las arquitecturas actuales se pueden unificar **si todos los threads de medio warp acceden al mismo segmento de memoria**, independientemente del patrón de acceso.

- Si los threads acceden a N segmentos distintos de memoria, entonces se producen N transacciones.

Memoria Global

- El tamaño del segmento puede ser:
  - 32 bytes si todos los *threads* acceden a palabras de 1 byte,
  - 64 bytes si todos los *threads* acceden a palabras de 2 bytes,
  - 128 bytes si todos los *threads* acceden a palabras de 4 bytes.

- Si los threads no acceden a todos los datos del segmento, entonces se leen datos que no se usan, desperdiciando ancho de banda.
- El hardware facilita un sistema para acotar la cantidad de datos a traer dentro del segmento, pudiendo traer subsegmentos de 32 bytes o 64 bytes.

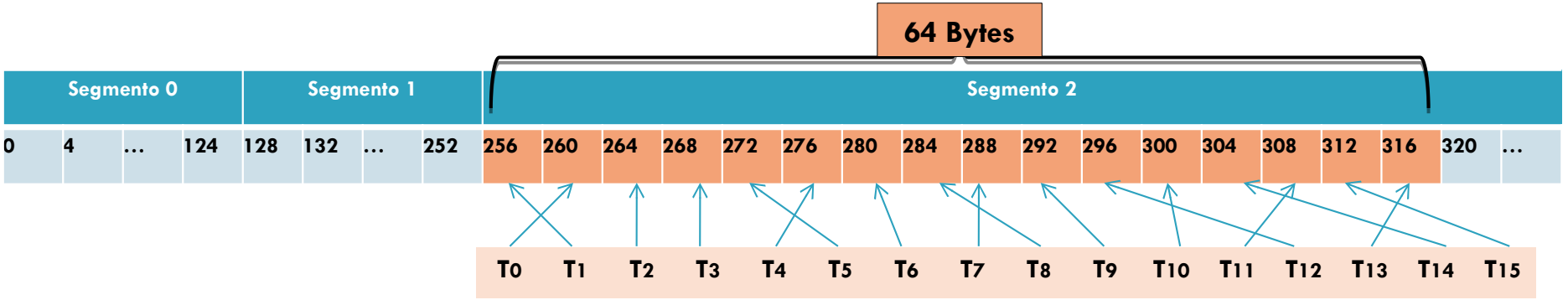
Memoria Global

- Suponer una memoria global:
  - Palabras de 4 Bytes.
  - Segmentos de 128 Bytes.

Segmento 0 - direcciones (0 a 127)					Segmento 1 - direcciones (128 a 255)					Segmento 2 - direcciones (256 a 383)				
0	4	8	…	124	128	132	136	…	252	256	260	264	…	380

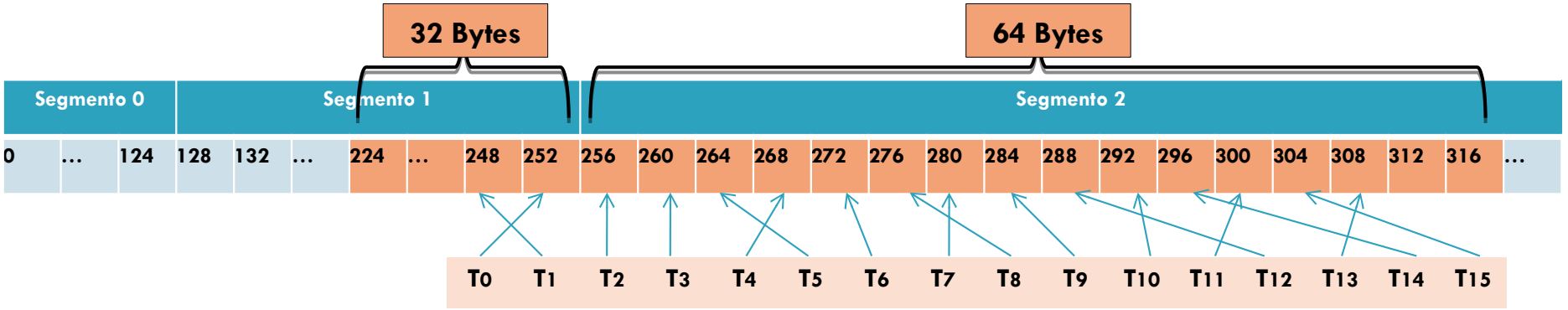
Memoria Global

- Ejemplo1:
  - Los threads acceden a 16 posiciones consecutivas alineadas con un segmento de 128Bytes.
  - Se produce una sola transacción y el hardware trae solo 64Bytes para evitar leer datos innecesarios.



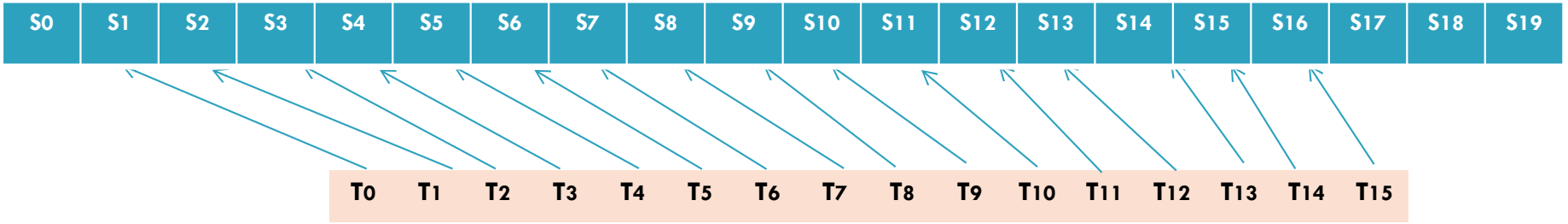
Memoria Global

- Ejemplo2:
  - Los threads acceden a 16 posiciones alojadas en distintos segmentos de 128Bytes.
  - Se generan dos transacciones (32 y 64 Bytes).



Memoria Global

- Ejemplo3:
  - Los threads acceden a 16 posiciones, cada una alojada en distintos segmentos de 128Bytes.
  - Se generan 16 transacciones.



Agenda

I. *CUDA Capability*

II. *Características generales de la jerarquía de memoria Nvidia*

III. ***Descripción de las distintas memorias de la jerarquía NVidia***

   I. *Memoria global*

   II. ***Memoria de constantes***

   III. *Memoria de texturas*

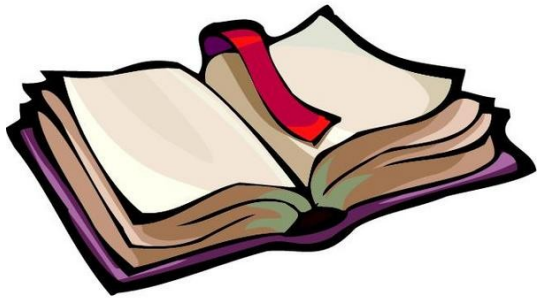
   IV. *Memoria compartida*

   V. *Registros*

   VI. *Memoria local*

IV. *Resumen*

V. *Limitaciones*



Memoria de Constantes

- **Visibilidad:** sólo lectura desde GPU. Escritura desde CPU.
- **Velocidad:** es una memoria off-chip, por lo tanto su acceso es lento. Aunque mejorado por cache on chip.
- **Variables:**
  - **Alcance:** *Todos los threads de un grid y/o de una aplicación.*
  - **Tiempo de vida**: toda la aplicación.

Memoria de Constantes

- La declaración de una variable en la memoria constante de la GPU la realiza el host y debe estar precedida por la palabra clave __constant__.

- Opcionalmente se le puede agregar la palabra clave __device__ antes de __constant__ y se logra el mismo efecto (La diferencia está en que se carga su valor con la función cudaMemcpyToSymbol).

Memoria de Constantes

- Con los patrones de acceso adecuados, el acceso a la memoria de constantes es rápido y paralelo.
- El tamaño total de la memoria de constante para una aplicación se limita a 64KB (Dependiendo de la arquitectura - capability).
- El límite de la caché para esta memoria es de 8KB o 10KB (Dependiendo de la arquitectura - capability).

Memoria de Constantes

- Los costos de acceso pueden ser:
  - Semejante al acceso a registro: si la referencia está en cache y uno o más threads del medio warp acceden a la misma posición.
  - Ligeramente mayor a un acceso a registro si los threads del medio warp acceden a posiciones distintas de la cache.
  - Igual de costoso a memoria global si se produce un fallo de cache.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. **Descripción de las distintas memorias de la jerarquía NVidia**

   I. Memoria global

   II. Memoria de constantes

   III. **Memoria de texturas**

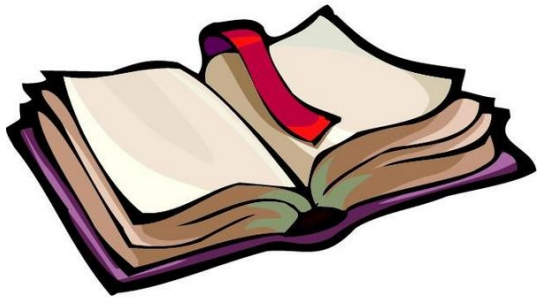
   IV. Memoria compartida

   V. Registros

   VI. Memoria local

IV. Resumen

V. Limitaciones



Memoria de texturas

- **Visibilidad:** sólo lectura desde GPU. Escritura desde CPU.
- **Velocidad:** es una memoria off-chip, por lo tanto su acceso es lento. Aunque mejorado por cache on chip.
- **Variables:**
  - **Alcance:** *Todos los threads de un grid y/o de una aplicación.*
  - **Tiempo de vida**: toda la aplicación.

Memoria de texturas

- El costo de acceso es distinto a las otras memorias, está optimizada para aprovechar la localidad espacial de dos dimensiones, desde el punto de vista de las imágenes es muy probable que si se accede a una parte de ésta, se lea la imagen completa.

- Las lecturas en memoria de texturas tienen mayores beneficios que la memoria global y de constante para estructuras bi-dimensionales.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. **Descripción de las distintas memorias de la jerarquía NVidia**

   I. Memoria global

   II. Memoria de constantes

   III. Memoria de texturas

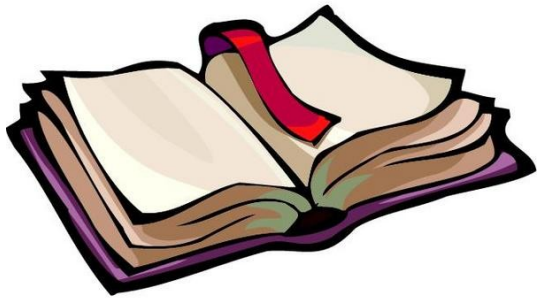
   IV. **Memoria compartida**

   V. Registros

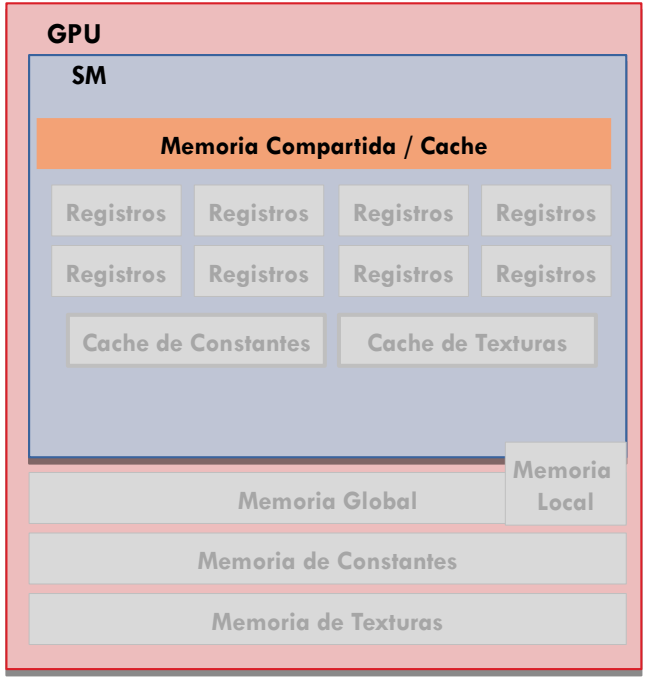
   VI. Memoria local

IV. Resumen

V. Limitaciones



Memoria compartida



- **Visibilidad:** lectura y escritura sólo desde GPU. Sin acceso desde CPU.
- **Velocidad:** es una memoria on-chip, por lo tanto su acceso es rápido.
- **Variables:**
  - **Alcance:** Todos los hilos de un bloque.
  - **Tiempo de vida:** un kernel.

Memoria Compartida

- La memoria compartida se organiza en bancos (generalmente de 1KB).
- Cada banco está formado por palabras sucesivas de 32bits.

Banco 0 (1KB)					Banco 1 (1KB)				
32 bits	32bits	32bits	…	32bits	32 bits	32bits	32bits	…	32bits

Memoria Compartida

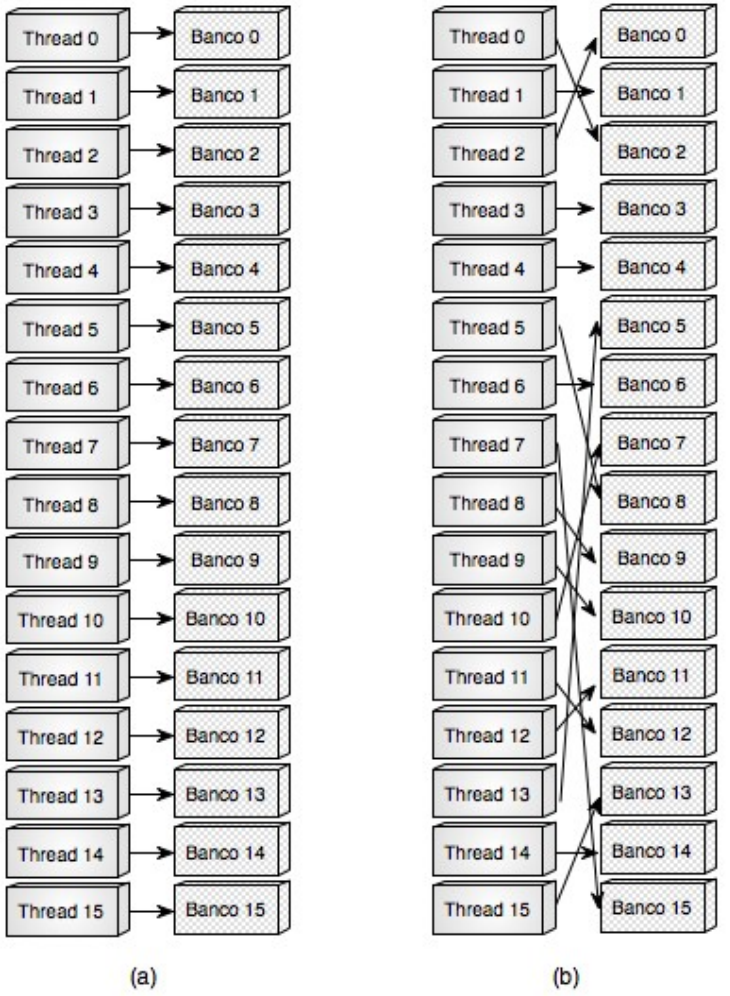
- Al igual que en la memoria global, los **accesos** a memoria compartida se resuelven de a **medio warp**.

- Para alcanzar el mejor rendimiento hay que **evitar** los **conflictos** de acceso a nivel de bancos en el mismo medio warp.

- Un **conflicto** implica **accesos** simultáneos de distintos threads a **direcciones diferentes del mismo banco** de memoria.

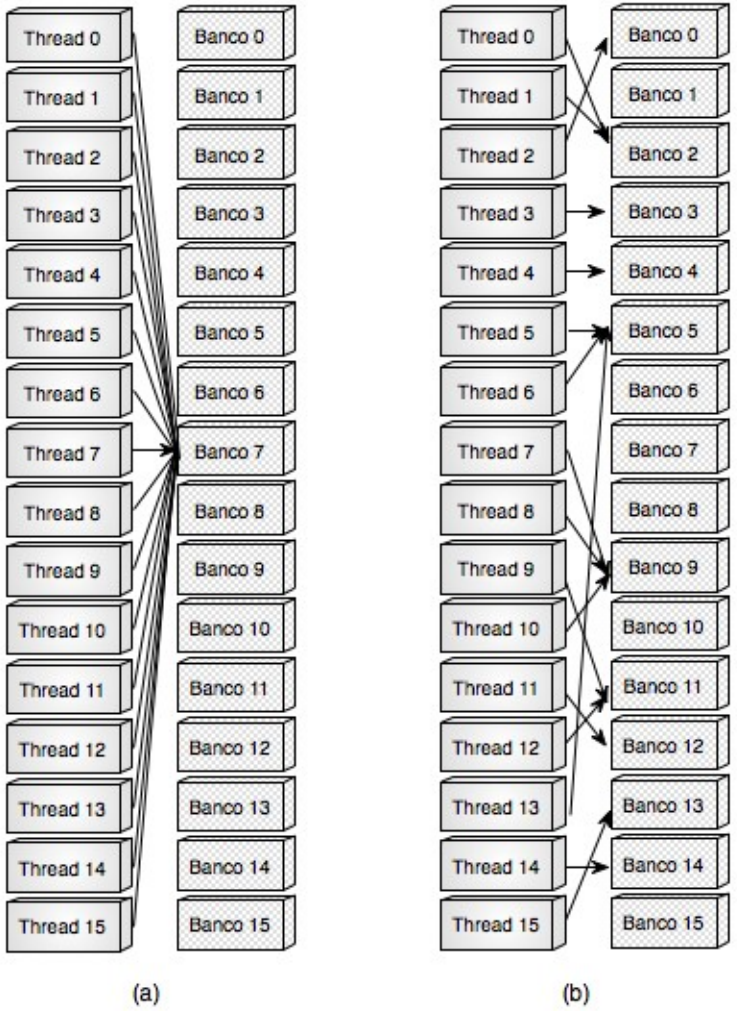
Memoria Compartida

- Ejemplo 1:
  - Accesos sin conflicto
  - *Todos los accesos se hacen a bancos diferentes*



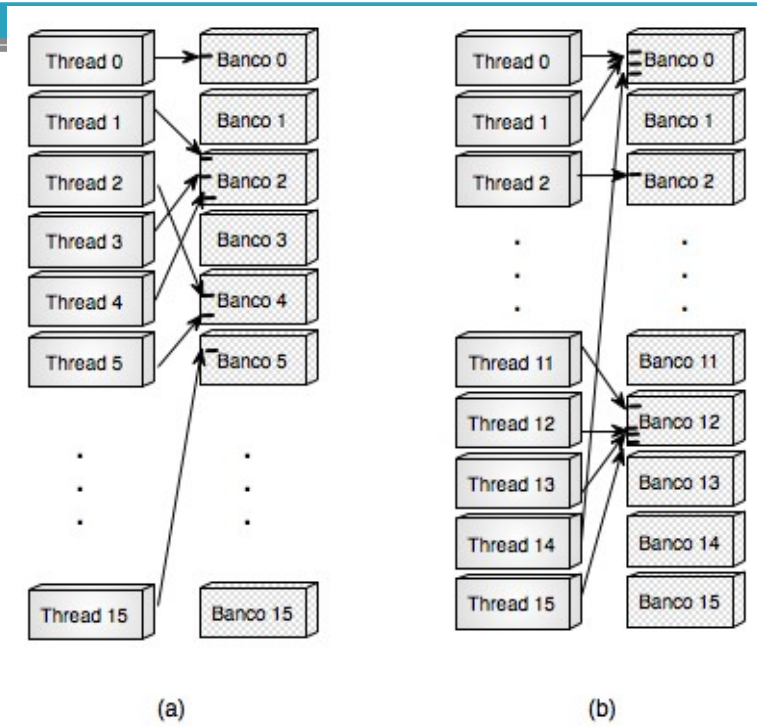
Memoria Compartida

- Ejemplo 2:
  - Accesos sin conflicto
  - Los accesos que coinciden en el mismo banco son a la misma dirección



Memoria Compartida

- Ejemplo 3:
  - Accesos con conflicto
  - Los accesos que *coinciden en el* mismo *banco son a direcciones diferentes*
  - *El hardware* serializa los accesos
    - Divide el acceso a memoria en tantos accesos libres de conflicto como sean necesarios
  - Existe pérdida del ancho de banda



Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. **Descripción de las distintas memorias de la jerarquía NVidia**

   I. Memoria global

   II. Memoria de constantes

   III. Memoria de texturas

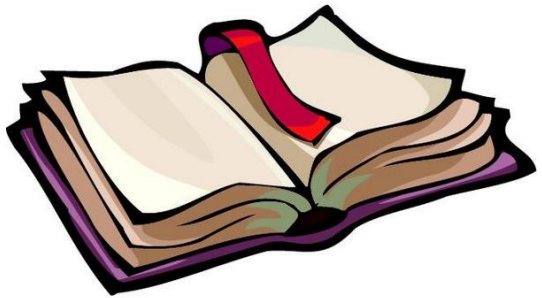
   IV. Memoria compartida

   V. **Registros**

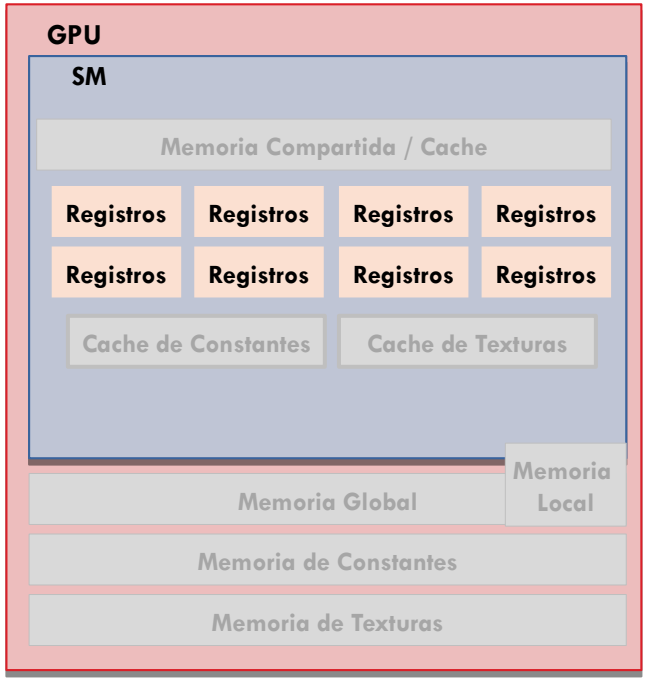
   VI. Memoria local

IV. Resumen

V. Limitaciones



Registros



- **Visibilidad:** lectura y escritura sólo desde GPU. Sin acceso desde CPU.
- **Velocidad:** es una memoria on-chip, por lo tanto su acceso es rápido. El acceso es considerado de costo cero y es altamente paralelo.
- **Variables:**
  - **Alcance:** se asignan a los threads individuales, teniendo cada uno la privacidad sobre ellos.
  - **Tiempo de vida:** un kernel.

Registros

- En registros se almacenan las variables escalares (NO arreglos), declaradas dentro de un kernel.
  - Por Ejemplo: los identificadores únicos de hilos
- Al invocarse un kernel, se crea una copia privada de cada una de las variables para cada thread

Registros

- Es importante tener en cuenta que los registros son **limitados.**
- **Ejemplo:** Un grid de 128 bloques de 256 threads, cada thread declara 8 variables privadas, entonces habría 32768 instancias de cada variable dentro del dispositivo. Se necesitarían 8*32768 = 262144 registros.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. **Descripción de las distintas memorias de la jerarquía NVidia**

   I. Memoria global

   II. Memoria de constantes

   III. Memoria de texturas

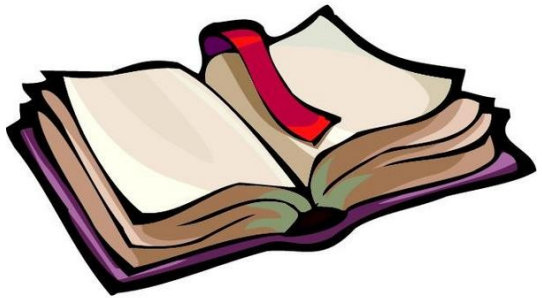
   IV. Memoria compartida

   V. Registros

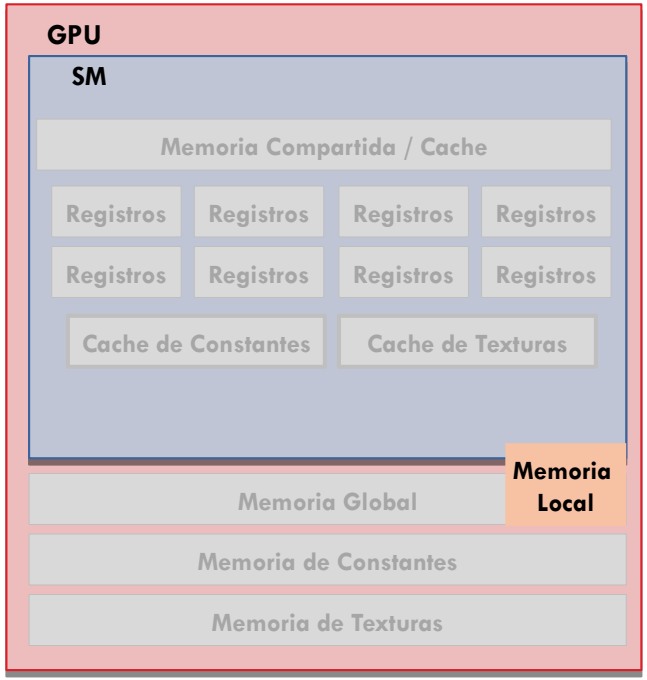
   VI. **Memoria local**

IV. Resumen

V. Limitaciones



Memoria local



- **Visibilidad:** lectura y escritura sólo desde GPU. Sin acceso desde CPU.
- **Velocidad:** El espacio de memoria local se encuentra dentro del espacio de memoria global y por lo tanto es costoso acceder.
- **Variables:**
  - **Alcance:** se asignan a los threads individuales, teniendo cada uno la privacidad sobre ellos.
  - **Tiempo de vida:** un kernel.

Memoria local

- La memoria local se utiliza para alojar variables:
  - Automáticas NO escalares (arreglos)
  - Automáticas escalares que no tienen lugar en los registros
- Esto implica mayor demora en su acceso.
- La declaración de una variable no escalar automática implica una instancia privada para cada uno de los threads ejecutando en el kernel, esto puede significar un alto consumo de la memoria global del dispositivo.

Agenda

I. *CUDA Capability*

II. *Características generales de la jerarquía de memoria Nvidia*

III. *Descripción de las distintas memorias de la jerarquía NVidia*

   I. *Memoria global*

   II. *Memoria de constantes*

   III. *Memoria de texturas*

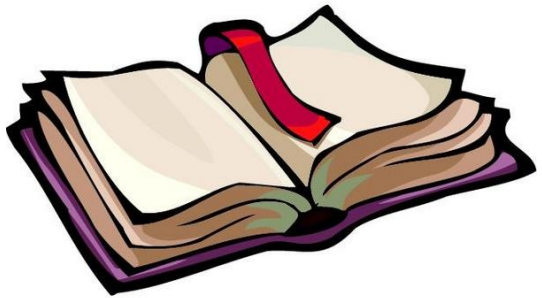
   IV. *Memoria compartida*

   V. *Registros*

   VI. *Memoria local*

IV. ***Resumen***

V. *Limitaciones*



Memoria – Resumen

Memoria	Ubicación	Cache	Acceso	Alcance	Tiempo de vida
Registro	On-Chip	No	Lectura y Escritura (Device)	Un hilo	Kernel
Local	Off-Chip	L1 y/o L2	Lectura y Escritura (Device)	Un hilo	Kernel
Compartida	On-Chip	No	Lectura y Escritura (Device)	Todos los hilos de un bloque	Kernel
Global	Off-Chip	L1 y/o L2	Lectura y Escritura (Device y Host)	Todos los hilos y el Host	Aplicación
Constantes	Off-Chip	Si	Lectura (Device) y Escritura (Host)	Todos los hilos y el Host	Aplicación
Texturas	Off-Chip	Si	Lectura (Device) y Escritura (Host)	Todos los hilos y el Host	Aplicación

Calificador	Memoria
Variables Automáticas escalares	De Registro
Variables Automáticas arreglos	Local
__*shared*__	Compartida
__constant__	De Constante
__device__ y punteros a memoria en GPU	Global

Agenda

I. *CUDA Capability*

II. *Características generales de la jerarquía de memoria Nvidia*

III. *Descripción de las distintas memorias de la jerarquía NVidia*

   I. *Memoria global*

   II. *Memoria de constantes*

   III. *Memoria de texturas*

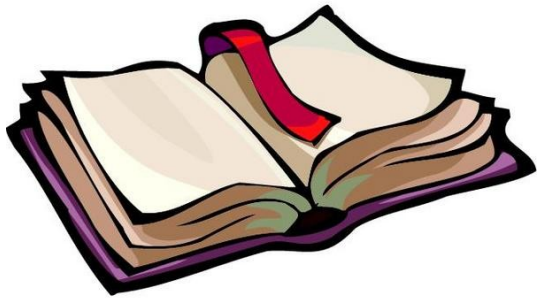
   IV. *Memoria compartida*

   V. *Registros*

   VI. *Memoria local*

IV. *Resumen*

V. ***Limitaciones***



Memoria como límite al paralelismo

- Aunque existen memorias muy rápidas (registro, compartida y constantes), su capacidad es muy limitada comparadas con la memoria global.

- Si los bloques tienen demasiados hilos, la capacidad que cada hilo puede usar de memoria compartida o de registro se limita.

Memoria como límite al paralelismo

- Ejemplo: Arquitectura G80, cada bloque puede tener 765 hilos, pero 10 registros cada uno. Si cada hilo necesita más de 10 registros CUDA reduce la concurrencia.

- En la misma arquitectura, si se tiene una memoria compartida de 16kB que se comparte entre todos los bloques que soporta un SM (8 para G80), entonces cada bloque podrá usar 2KB. Si necesitan más de 2KB cada uno entonces el SM se limita a ejecutar tantos bloques como recursos tenga, si necesitan 5KB solo atenderá 3 bloques.