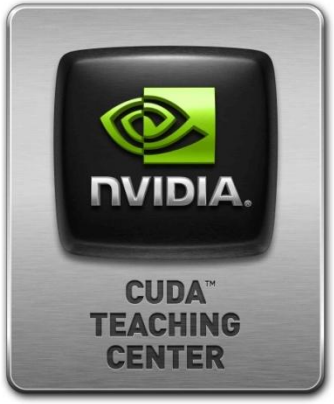
Programación masivamente paralela en procesadores gráficos (GPUs)

E. Dufrechou, M. Freire, P. Ezzatti y M. Pedemonte



Clase 4
Arquitectura de la GPU

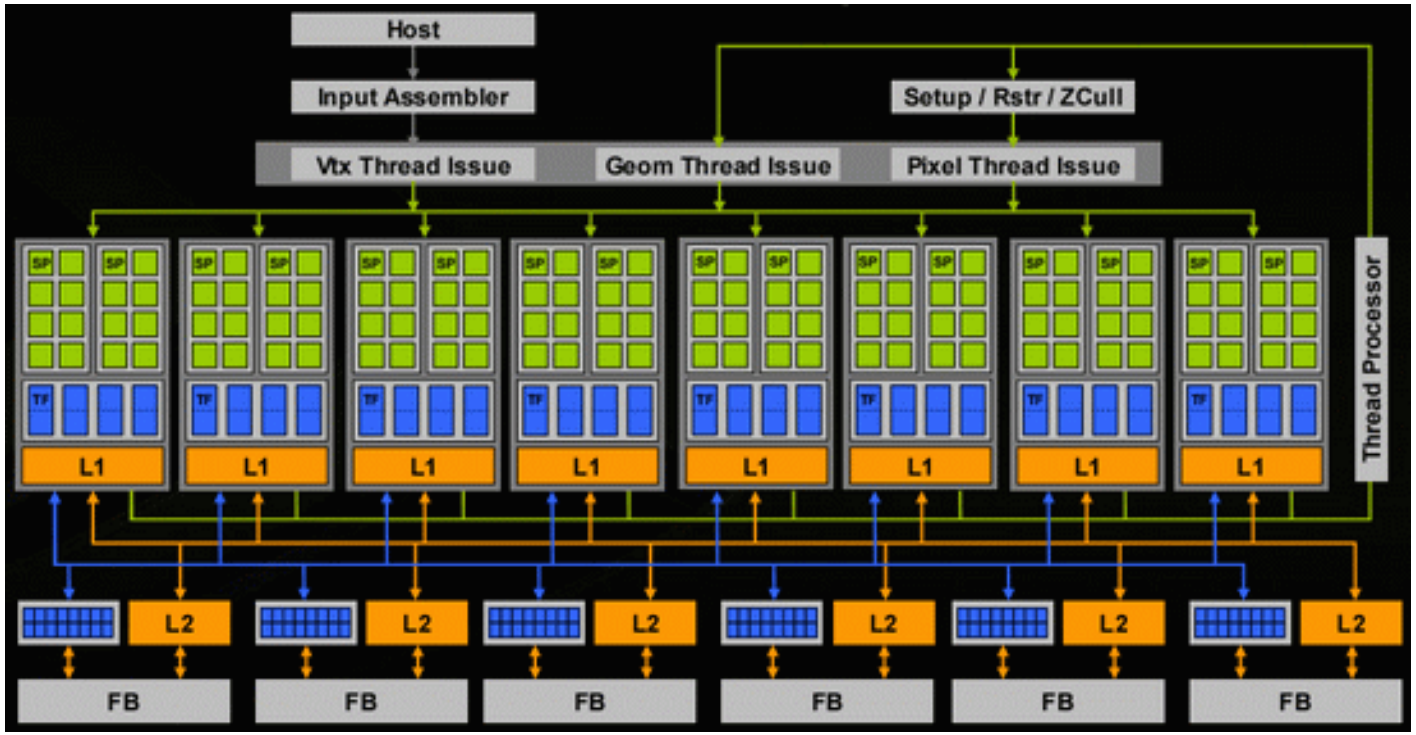
Contenido

- Arquitectura CUDA – G80
- Modelo de Ejecución
- Jerarquía de Memoria
- Arquitectura CUDA – GT200
- Arquitectura CUDA – Fermi
- Arquitectura CUDA – Kepler
- Compute Capabilities

Arquitectura CUDA – G80

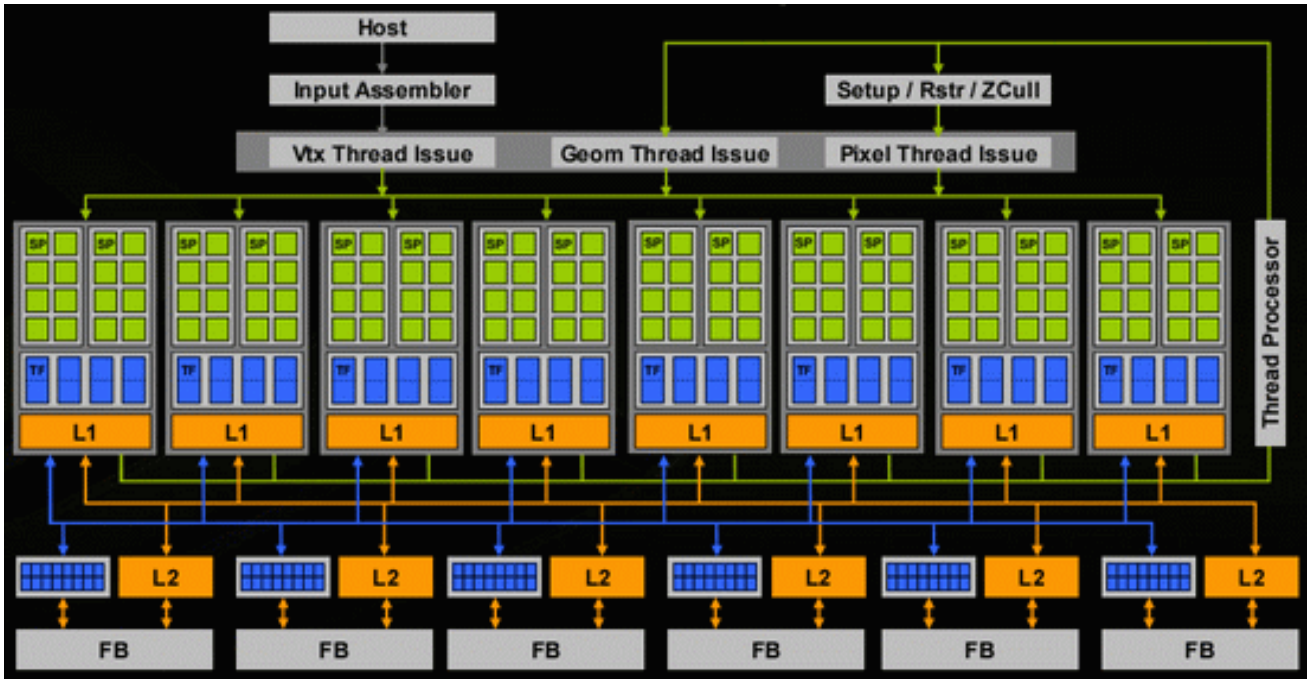
Arquitectura CUDA – G80

La Arquitectura G80



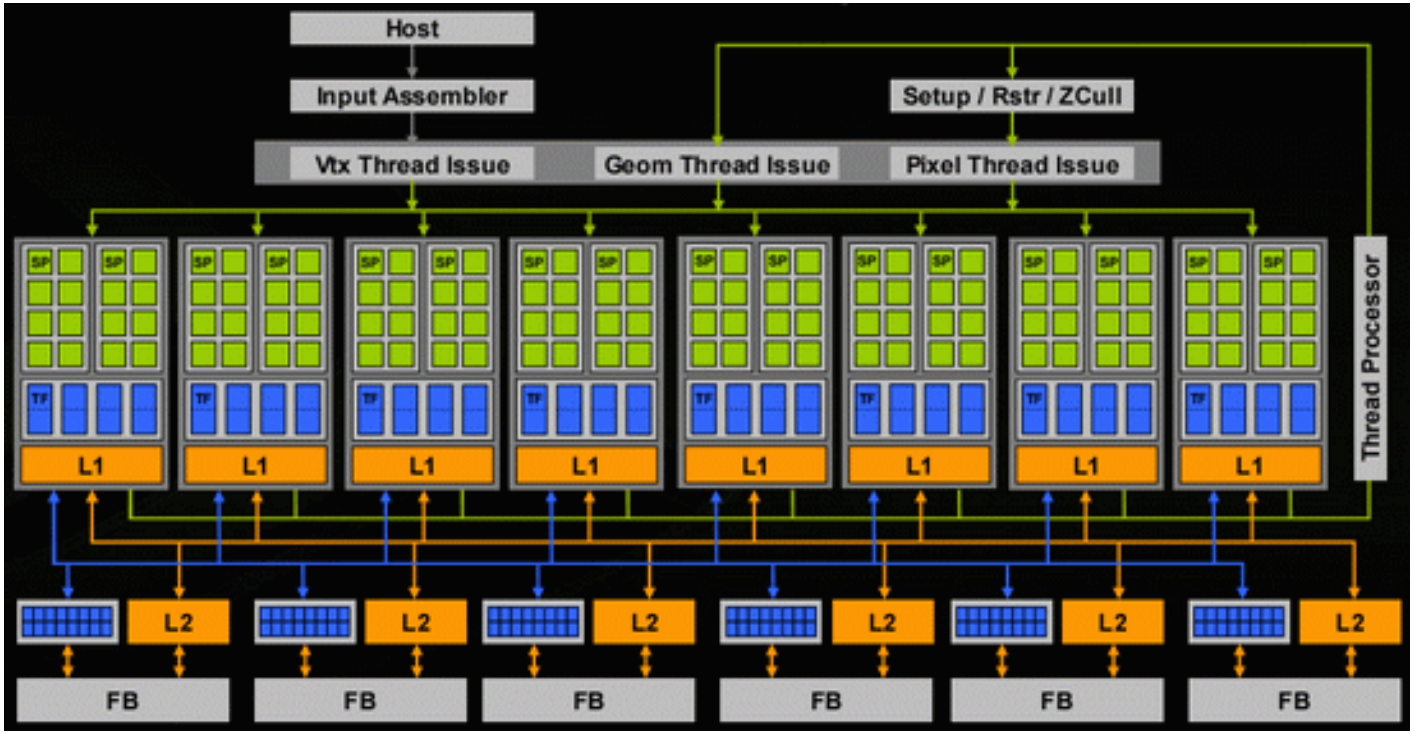
Arquitectura CUDA – G80

- La arquitectura CUDA puede verse como un arreglo escalable de multiprocesadores.
- SMs: multithreaded Streaming Multiprocessors.
- La arquitectura de la G80 (primera de CUDA) tiene 16 SMs.



Arquitectura CUDA – G80

- Los SMs se agrupan en pares formando 8 Thread Processing Clusters (TPCs).
- Los TPCs contienen los caches de texturas y de memoria constante que son compartidos por los SMs.



Arquitectura CUDA – G80

- **Un multiprocesador (SM) de la arquitectura G80 consiste en:**
  - **ocho procesadores escalares (Streaming Processors, SP). También conocidos como CUDA cores.**
  - **unidad de instrucciones multihilo**
  - **chip de memoria compartida**
  - **dos unidades especiales (Special Function Units, SFU) para computar operaciones trascendentales como sin, cos, log, y sqrt.**

Arquitectura CUDA – G80

- Cada CUDA core tiene una unidad que permite realizar una operación multiply-add y una unidad que permite hacer una multiplicación.
- Es decir que los CUDA cores básicamente son una ALU.
- Los CUDA cores no tienen registros propios o caches.
- Los registros se manejan a nivel de multiprocesador.
- Los CUDA cores soportan la aritmética de punto flotante de simple precisión (32 bits) definida en el estándar IEEE 754-1985.

Arquitectura CUDA – G80

- La arquitectura de la CPU utiliza caché de datos para ocultar la latencia en el acceso a la memoria RAM.
- La arquitectura de la GPU elimina (o reduce sensiblemente) el caché de datos.
- Las GPUs para ocultar la latencia en el acceso a la memoria:
  - cuando un conjunto de threads accede a memoria, se suspende su ejecución hasta que los datos hayan llegado al SM.
  - otro conjunto de threads comienza a ejecutar.
  - el scheduling (planificación) se realiza por hardware en forma transparente para el programador y prácticamente sin costo.
- Esto permite tener muchos más threads que unidades de ejecución.

Arquitectura de CUDA – G80

- **Los multiprocesadores tienen la capacidad de crear, gestionar y ejecutar hilos concurrentemente prácticamente sin costo de planificación.**
- **Los SMs de la arquitectura G80 pueden ejecutar hasta 768 hilos concurrentemente pero no al mismo momento.**
- **Por lo tanto, la tarjeta puede ejecutar 12288 hilos concurrentemente!!!**

Modelo de Ejecución Clásico

Modelo de ejecución clásico

- La arquitectura CUDA permite abstraer la GPU como un conjunto de multiprocesadores compuestos a su vez por un conjunto de procesadores orientados a la ejecución de hilos.
- En el modelo de ejecución de CUDA cada multiprocesador ejecuta el mismo programa pero sobre distintos datos.
- Los distintos multiprocesadores NO tiene que estar ejecutando la misma instrucción en el mismo momento.
- Este paradigma de programación se conoce con el nombre SPMD (single program multiple data).

Modelo de ejecución clásico

- **Internamente cada multiprocesador sigue el paradigma de programación paralela SIMT (Single Instruction Multiple Threads).**
- **SIMT es similar a SIMD (Single Instruction Multiple Data) ya que una sola instrucción controla múltiples elementos de procesamiento.**
- **Sin embargo, no son exactamente la misma cosa.**

Modelo de ejecución clásico

- **SIMD (como las vector machines o Intel MMX y SSE):**
  - **Existe una única unidad de procesamiento que ejecuta secuencialmente instrucciones.**
  - **Algunas instrucciones son escalares y otras vectoriales, pero en ese caso es una instrucción operando sobre varios elementos de datos.**

Modelo de ejecución clásico

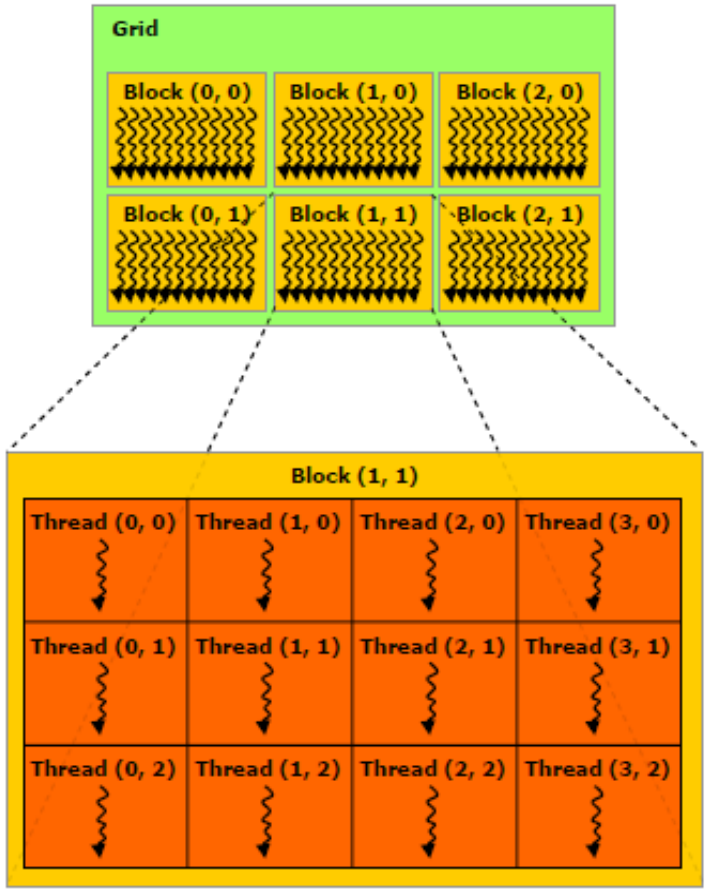
- **SIMT (como las GPUs):**
  - **Cada "carril" del cómputo vectorial puede verse como un hilo separado.**
  - **Existe una única unidad (hardware sequencer) que se encarga de gestionar el trabajo en paralelo sobre un grupo de esos hilos.**
  - **Una única instrucción se difunde a todos los elementos de procesamiento.**
  - **Si todos los hilos deben ejecutar la misma instrucción, se dice que los hilos mantienen coherencia.**
  - **Si los hilos deben ejecutar instrucciones distintas, los hilos divergen.**
  - **La unidad (hardware sequencer) mantiene la información de que hilos han divergido.**

Modelo de ejecución clásico

- **SIMT (como las GPUs):**
  - **La unidad difunde una única instrucción por ciclo a todos los elementos de procesamiento que pueden ejecutar (en función de las divergencias). Algunos hilos se deshabilitan (los que divergieron).**
- **Como consecuencia del funcionamiento descrito, el mismo programa podría ejecutar si cada hilo ejecutara en forma independiente en un core distinto.**
- **En contraste con SIMD, SIMT permite tener:**
  - **paralelismo a nivel de datos: cuando los hilos son coherentes.**
  - **paralelismo a nivel de hilo: cuando los hilos divergen, cada hilo ejecuta en forma independiente.**

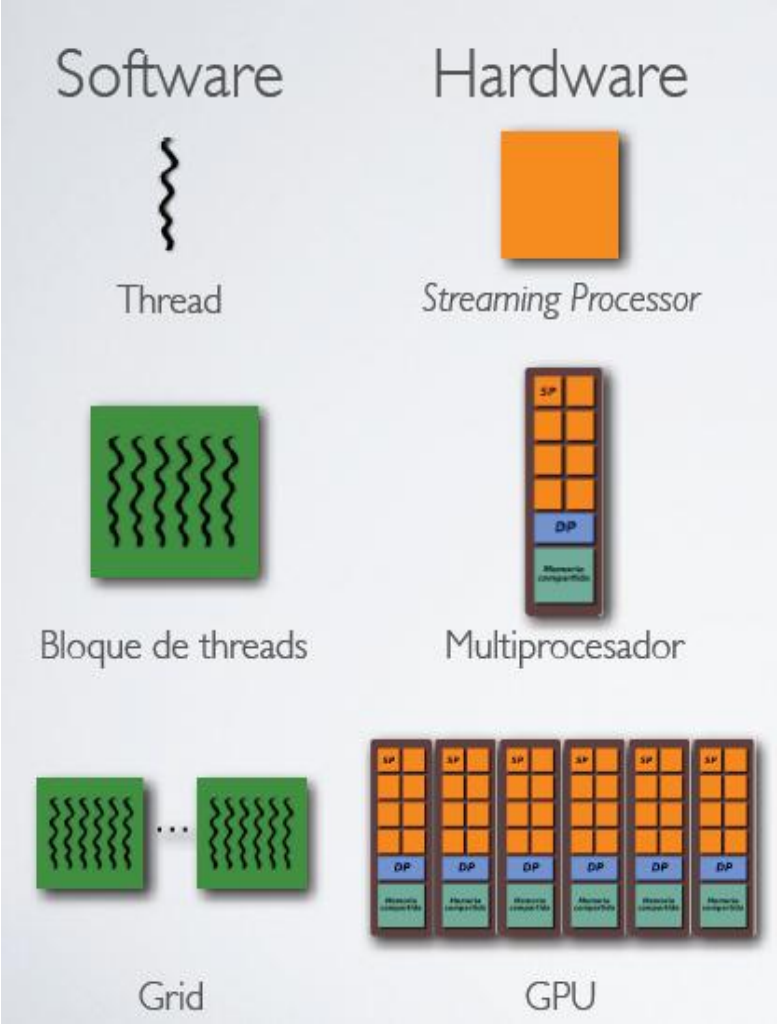
Modelo de ejecución clásico

- Los programas que ejecutan en la GPU se denominan kernels.
- La GPU puede ejecutar un kernel a la vez.
- La ejecución de un kernel es realizada por hilos que se organizan en bloques.
- Los bloques pueden ser unidimensionales, bidimensionales o tridimensionales.
- Los bloques se organizan en un grid.
- El grid puede ser unidimensional o bidimensional.



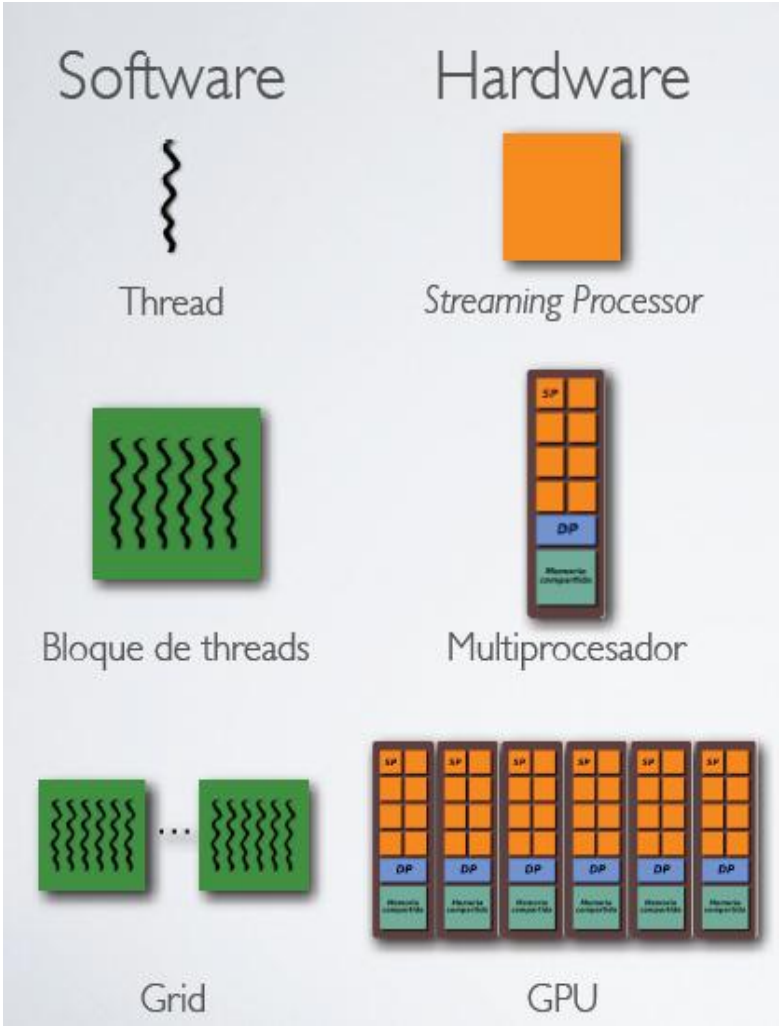
Modelo de ejecución clásico

- **Cada bloque es un subconjunto de los cómputos que será ejecutado en un multiprocesador (MP) en forma independiente.**
- **Si hay suficientes MPs disponibles, todos los bloques de un grid son ejecutados en paralelo.**



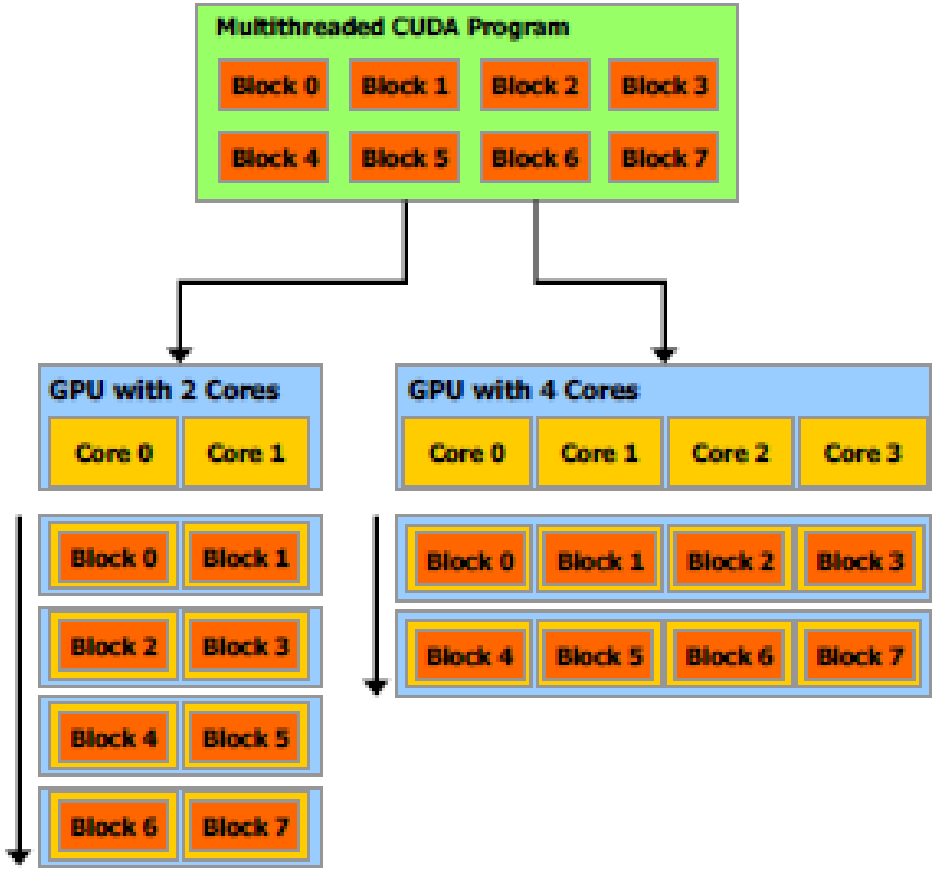
Modelo de ejecución clásico

- **En caso contrario, el scheduler gestiona la ejecuciones de los bloques:**
  - **Un MP puede ejecutar múltiples bloques.**
  - **El orden de ejecución de los bloques no es conocido a priori.**
- **Cada hilo de un bloque ejecuta en un CUDA core del MP.**



Modelo de ejecución clásico

- **Organizar la ejecución de esta manera permite escalar automáticamente.**



Modelo de ejecución clásico

- La ejecución se planifica en base a warps.
- El MP crea, gestiona, planifica y ejecuta en base a grupos de hilos llamados warps.
- El tamaño de warp en la arquitectura G80 es 32 hilos.
- Cuando un MP debe ejecutar un bloque, lo particiona en warps.
- Cada warp contiene hilos consecutivos.

Modelo de ejecución clásico

- Un warp ejecuta una instrucción a la vez.
- Si los hilos de un warp divergen debido a una condición de bifurcación dependiente de los datos:
  - El warp serializa la ejecución de cada camino de la bifurcación, deshabilitando los hilos que no forman parte del camino de ejecución.
  - Cuando todos los caminos completan su ejecución, los hilos convergen al mismo camino de ejecución.
- Como consecuencia, la eficiencia máxima se logra cuando los hilos de un warp coinciden en su camino de ejecución.
- La divergencia ocurre únicamente dentro de un warp.
- Distintos warps ejecutan en forma independiente, sin tener en cuenta si están ejecutando caminos comunes o disjuntos.

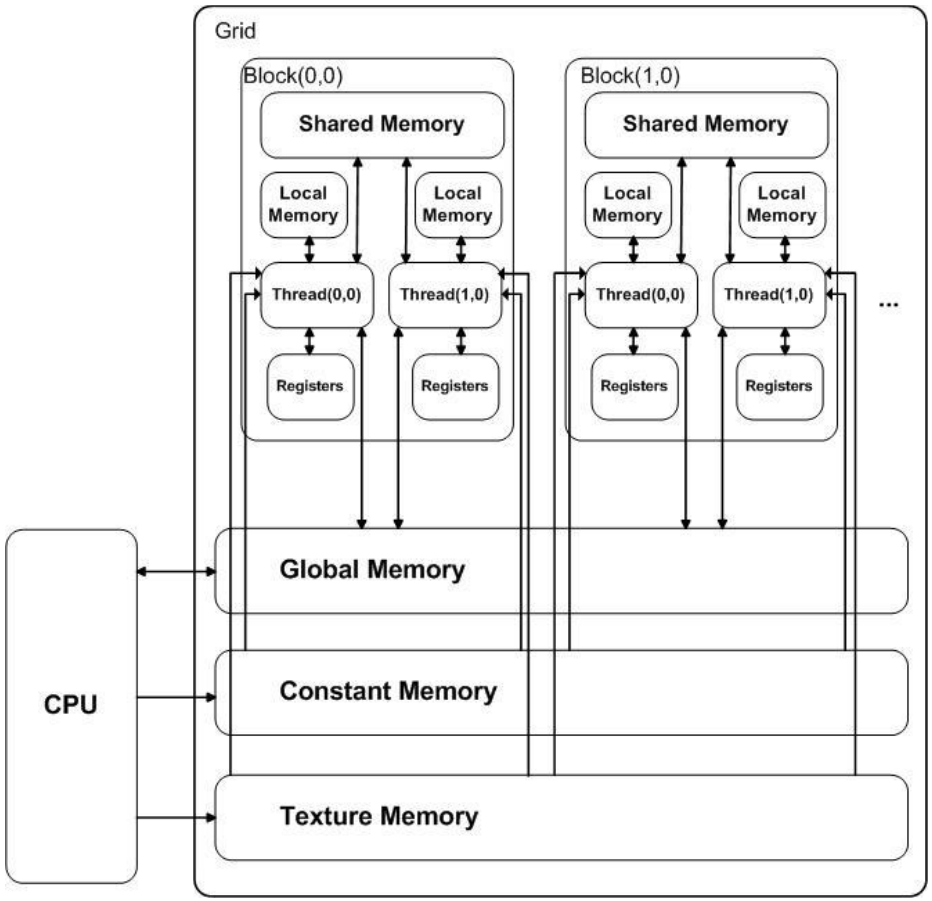
Jerarquía de Memoria

Jerarquía de memoria

- **Existen varios tipos de memoria distintos en el dispositivo:**
  - **Global**
  - **Local**
  - **Compartida**
  - **Registros**
  - **Constante**
  - **Texturas**
- **Diremos que una memoria es on-chip cuando está en el mismo chip que los CUDA cores.**
- **Diremos que una memoria es off-chip cuando NO está en el mismo chip que los CUDA cores.**

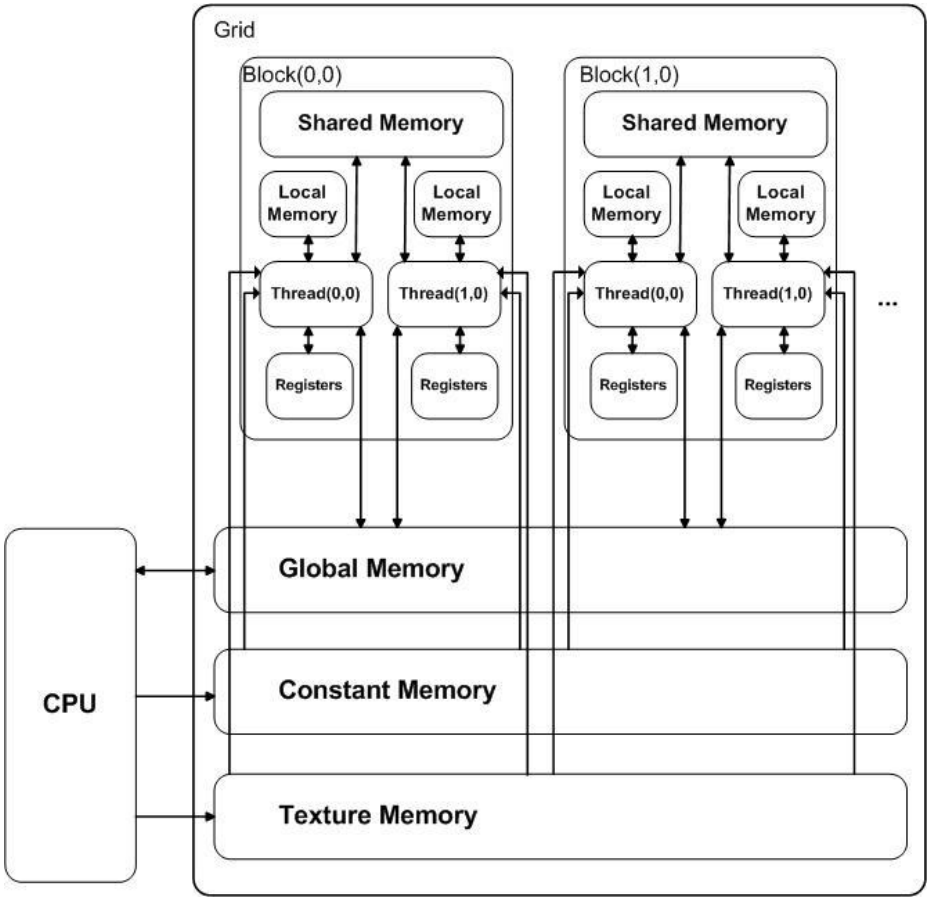
Jerarquía de memoria

- **Memoria global:**
  - Todos los hilos que ejecutan en la GPU tienen acceso al mismo espacio global de memoria.
  - Es off-chip.
  - El acceso es el más lento y no es cacheado.
- **Registros:**
  - Es on-chip.
  - Es la más rápida de la GPU.
  - Sólo son accesible por cada hilo.
  - Este espacio de memoria es gestionada por el compilador.



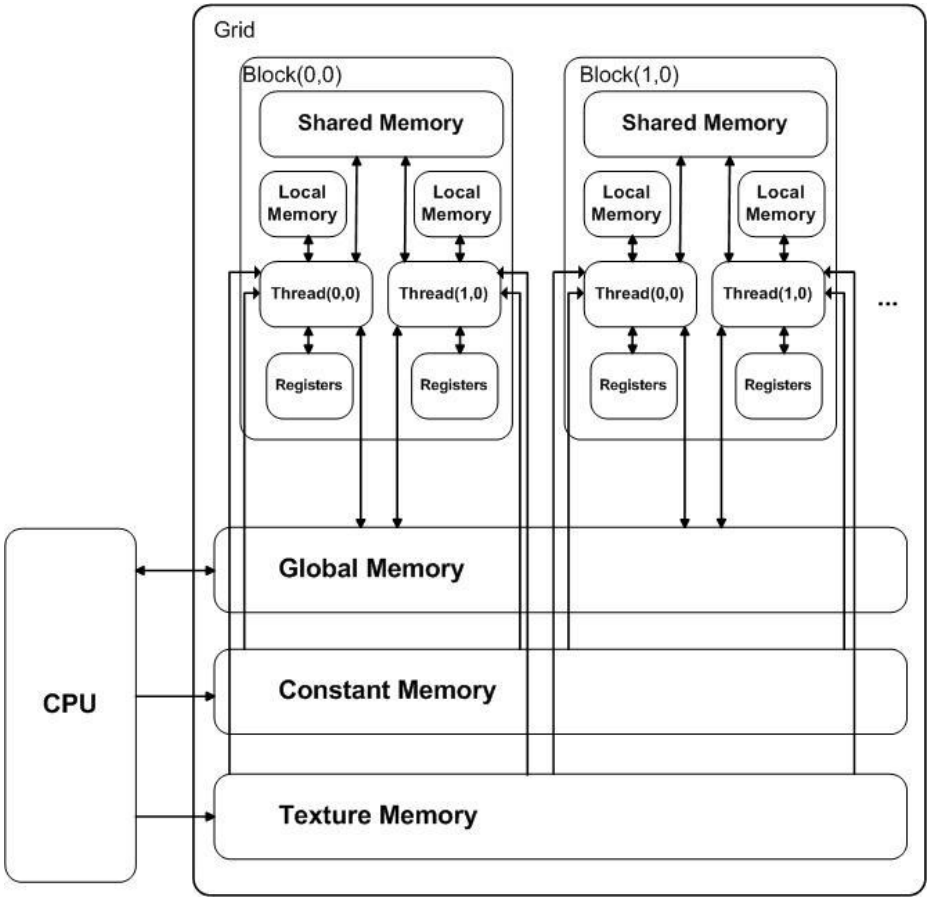
Jerarquía de memoria

- **Memoria local:**
  - **Cada hilo tiene su propia memoria local.**
  - **Es una de las memorias más lentas de la GPU y no es cacheado.**
  - **Es off-chip.**
  - **Este espacio de memoria es gestionada por el compilador.**
  - **La figura muestra esta memoria cercana a los hilos y como privada a cada hilo. Sin embargo, realmente es off-chip.**



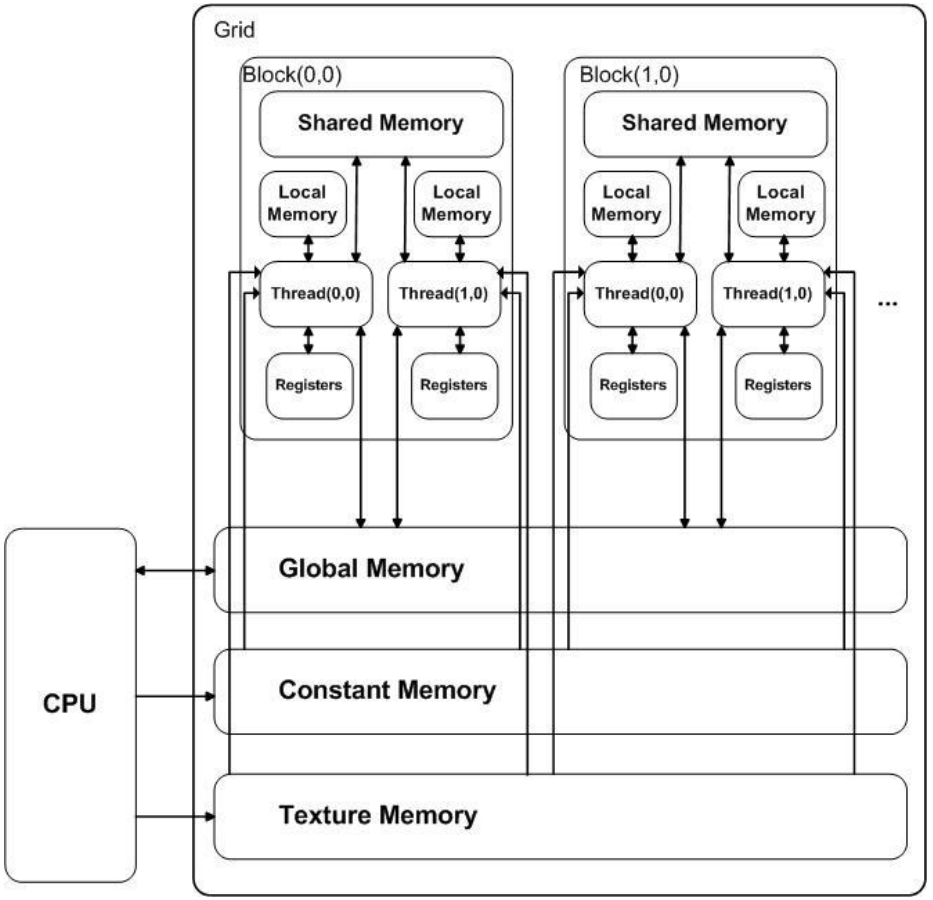
Jerarquía de memoria

- **Memoria compartida:**
  - **Cada bloque tiene un espacio de memoria compartida que es prácticamente tan rápida como los registros.**
  - **Puede ser accedida por cualquier hilo del bloque.**
  - **Es on-chip.**
  - **Su tiempo de vida es igual al tiempo de vida del bloque.**



Jerarquía de memoria

- **Memoria constante:**
  - Es una memoria rápida.
  - Para el dispositivo es solamente de lectura
  - Es off-chip aunque es cacheada.
  - Puede ser vista como un caché a memoria global más que como un espacio de memoria distinto.
- **Texturas:**
  - Tiene características similares a la memoria constante.



Jerarquía de memoria

Memoria	Ubicación	Caché	Acceso	Alcance	Existencia
Constante	Off-chip	Sí	R	Device	Aplicación
Local	Off-chip	No	R-W	Thread	Aplicación
Global	Off-chip	No	R-W	Device	Aplicación
Compartida	Chip	No	R-W	Bloque	Bloque
Registros	Chip	-	R-W	Thread	Thread
Texturas	Off-chip	Sí	R??R-W??	Device	Aplicación

Jerarquía de memoria

- **Tamaño máximo de los espacios de memoria en la arquitectura G80:**
  - **Global: 768 MB – 1024 MB**
  - **Local: 16 KB**
  - **Compartida: 16 KB**
  - **Registros: 8 KB por multiprocesador**
  - **Constante: 64 KB (8 KB por TPC)**
  - **Texturas: 8 KB por TPC**

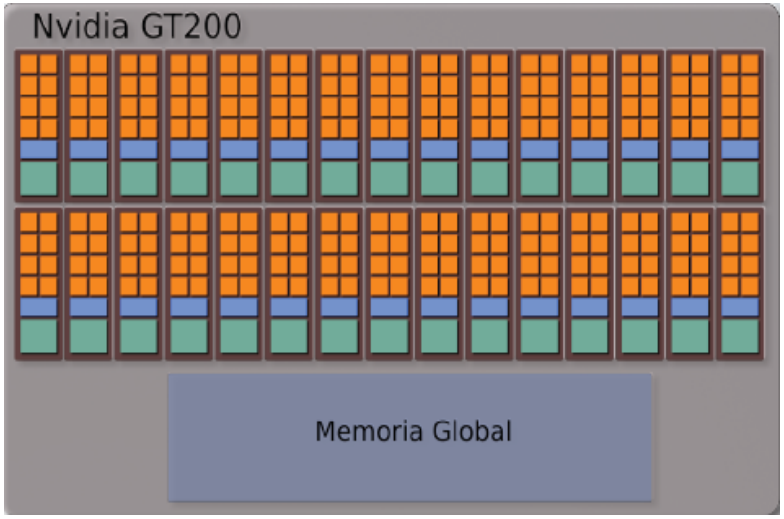
Arquitectura CUDA – GT200

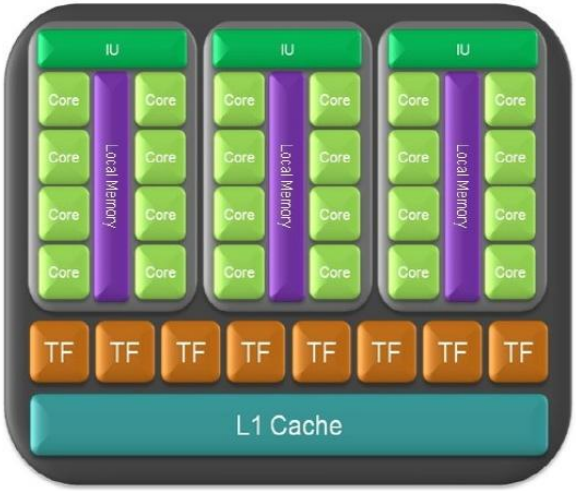
Arquitectura CUDA – GT200

- En 2008 se lanzó la segunda generación de CUDA.
- En realidad la arquitectura de la segunda generación es solamente una revisión de la primera generación.
- Algunos de los cambios más importantes son:
  - Soporte a aritmética de doble precisión (64 bits) del estándar IEEE 754-1985. Para ello cada MP incorpora una unidad de doble precisión.
  - El interfaz a memoria global se expande a 512 bits.
  - Soporte a operaciones atómicas read-modify-write en memoria compartida y memoria global.
  - Mejora en la performance de las SFUs.

Arquitectura CUDA – GT200

- **Algunos de los cambios más importantes son:**
  - **Los TPCs agrupan 3 SMs.**
  - **Las GPUs tienen 10 TPCs, resultando en 30 SMs y 240 CUDA cores.**
  - **El número de hilos que cada SM puede ejecutar en forma concurrente se incrementa a 1024 (30720 hilos por GPU).**





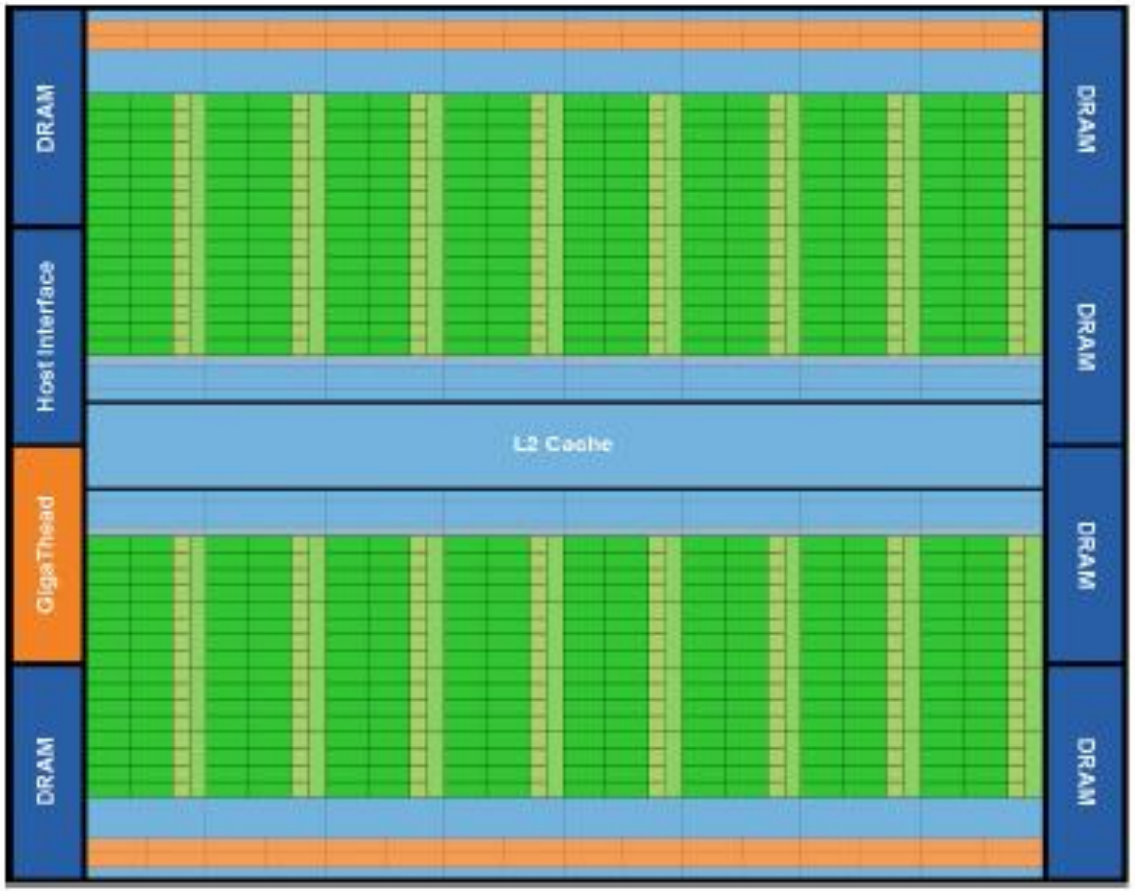
Arquitectura CUDA – Fermi

Arquitectura CUDA – Fermi

- En 2010 se lanza la tercera generación de CUDA.
- Esta nueva arquitectura representa una mejora sustancial sobre las arquitecturas previas.
- Algunos de los cambios más importantes son:
  - Los CUDA cores soportan completamente el estándar 754-2008 de la IEEE para simple y doble precisión (se agregó el soporte para números desnormalizados).
  - Se unifica el espacio de direcciones de memoria (global, shared y local), lo que permite dar soporte completo a C++.
  - Mejora en la performance de las operaciones atómicas.

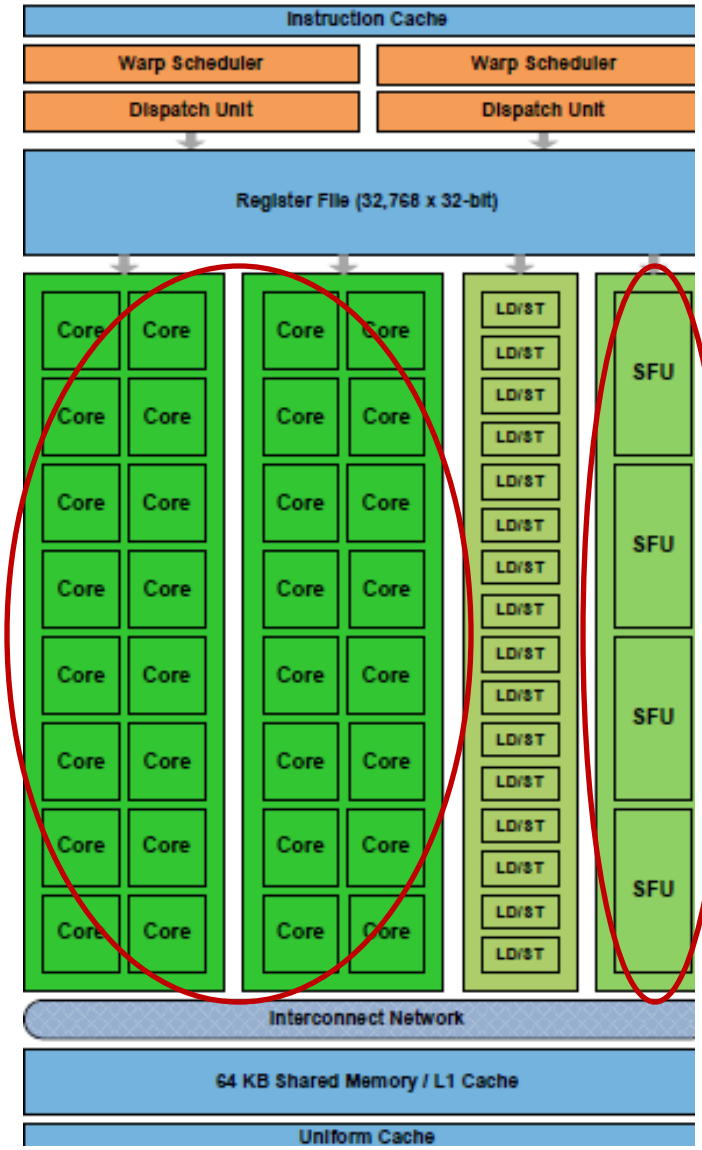
Arquitectura CUDA – Fermi

- **Algunos de los cambios más importantes son:**
  - **Tiene 16 SMs con 32 CUDA cores cada uno (512 cores en total).**



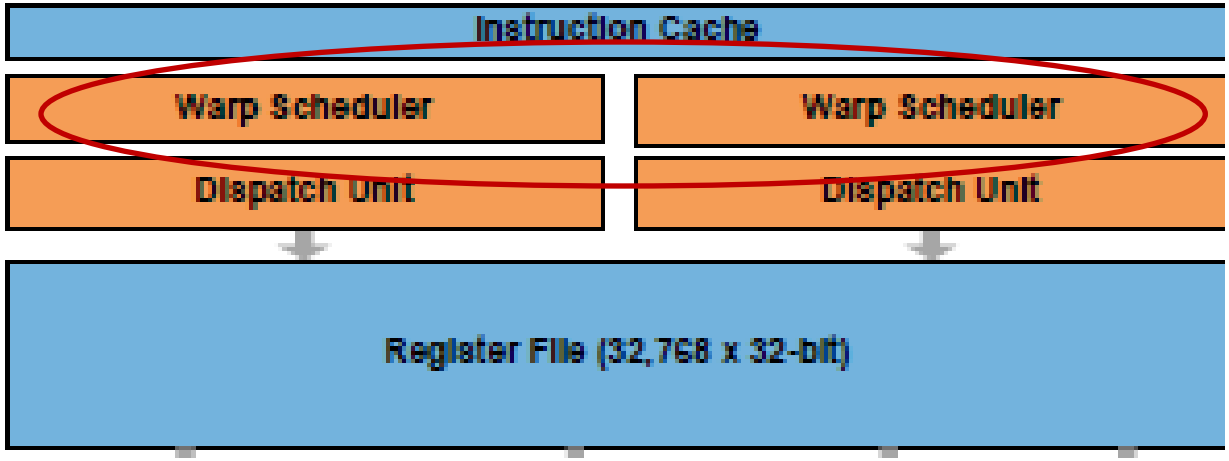
Arquitectura CUDA – Fermi

- **Algunos de los cambios más importantes son:**
  - **Los SMs están organizados en dos bloques de 16 cores cada uno.**
  - **Tienen 4 SFUs lo que permite que en ocho ciclos de reloj ejecute un warp.**
  - **El pipeline de SFUs está desacoplado de la dispatch unit por lo que se pueden despachar instrucciones a otras unidades mientras las SFUs están ocupadas.**
  - **Aumento significativo en la performance de punto flotante de doble precisión (relación 2x con simple precisión).**

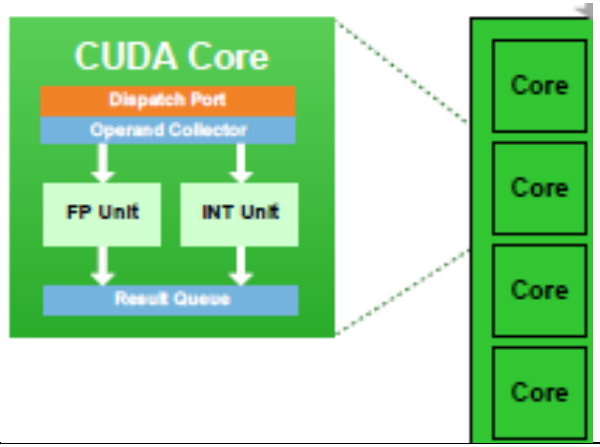


Arquitectura CUDA – Fermi

- **Algunos de los cambios más importantes son:**
  - **Cada SM tiene dos planificadores de warps (atienden dos warps a la vez, uno para bloque de cores) y una memoria en el chip de 64 KB.**

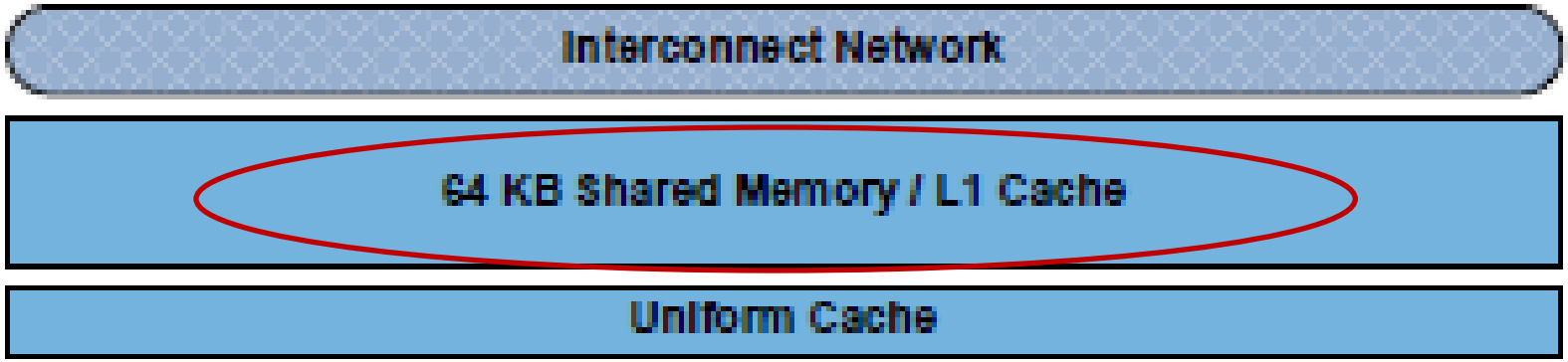


  - **Cada core se compone de una unidad para el procesamiento de punto flotante y una ALU.**



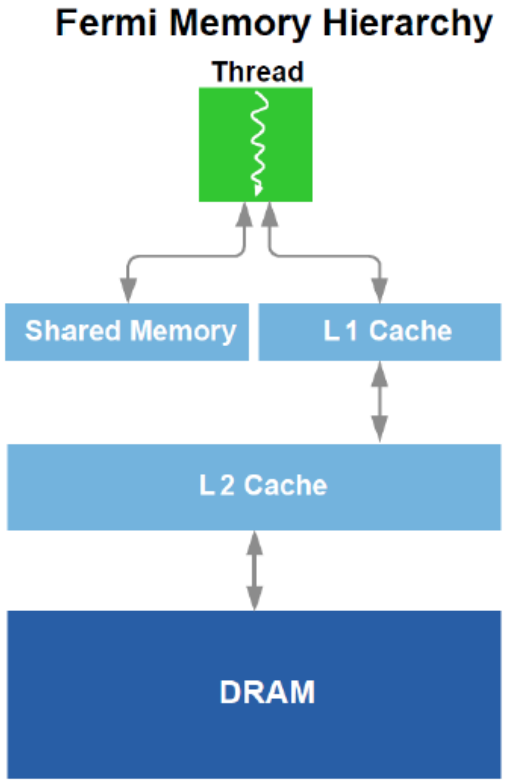
Arquitectura CUDA – Fermi

- **Algunos de los cambios más importantes son:**
  - **La memoria del chip funciona como un caché de primer nivel para la memoria global y una parte como memoria compartida.**
  - **Puede ser dividida 16K/48K o 48K/16K entre caché y memoria compartida.**



Arquitectura CUDA – Fermi

- **Algunos de los cambios más importantes son:**
  - **También se incorpora un caché de segundo nivel de 768 KB para el acceso a la memoria global.**



  - **Se incorporan mecanismos para la detección y corrección de errores en el acceso a datos.**

Arquitectura CUDA – Kepler

Arquitectura CUDA – Kepler

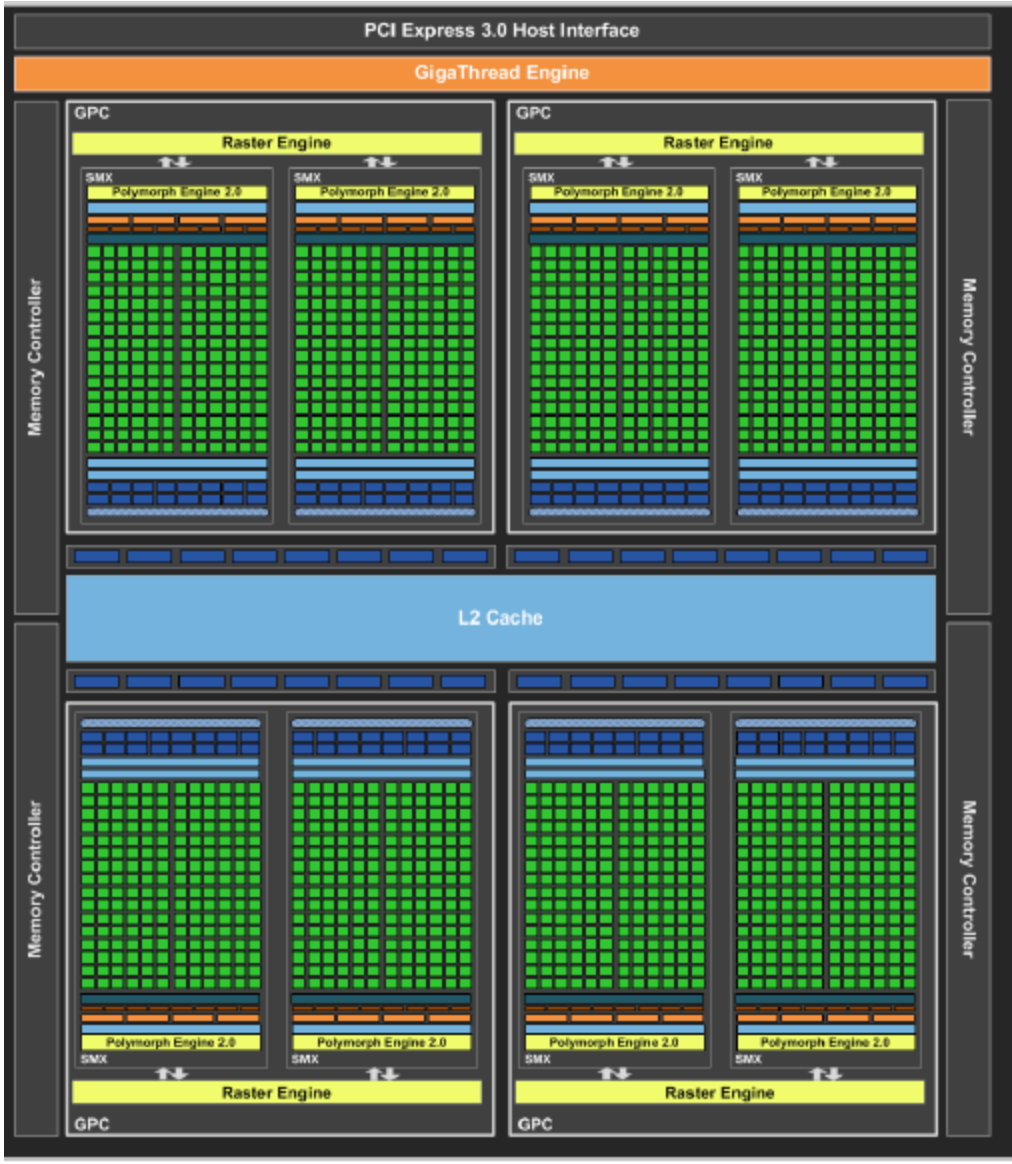
- En 2012 se lanza la cuarta generación de CUDA, conocida con el nombre de Kepler.
- Mejora significativa en la performance.
- Sin embargo, el foco está puesto en reducir el consumo de energía.
- El objetivo más importante del diseño de la arquitectura Kepler es obtener una mejor performance por watt.

Arquitectura CUDA – Kepler

- **Algunos cambios importantes son:**
  - **Los nuevos Streaming Multiprocessor se llaman SMX y funcionan a la frecuencia del reloj de los gráficos. Anteriormente los SMs funcionaban a la frecuencia de los shaders que eran aproximadamente 2x de los gráficos.**
  - **La reducción en la frecuencia es clave para la disminución en el consumo de energía:**
    - **Fermi GF110: Shader clock 1544 Mhz - Graphics clock 772 Mhz.**
    - **Kepler GK104: Graphics clock 1006 Mhz.**
  - **Pero entonces ¿cómo se logra mejorar la performance?**
  - **Aumentando la cantidad de cores!!!**
    - **Fermi GF110: 512 CUDA cores**
    - **Kepler GK104: 1536 CUDA cores**

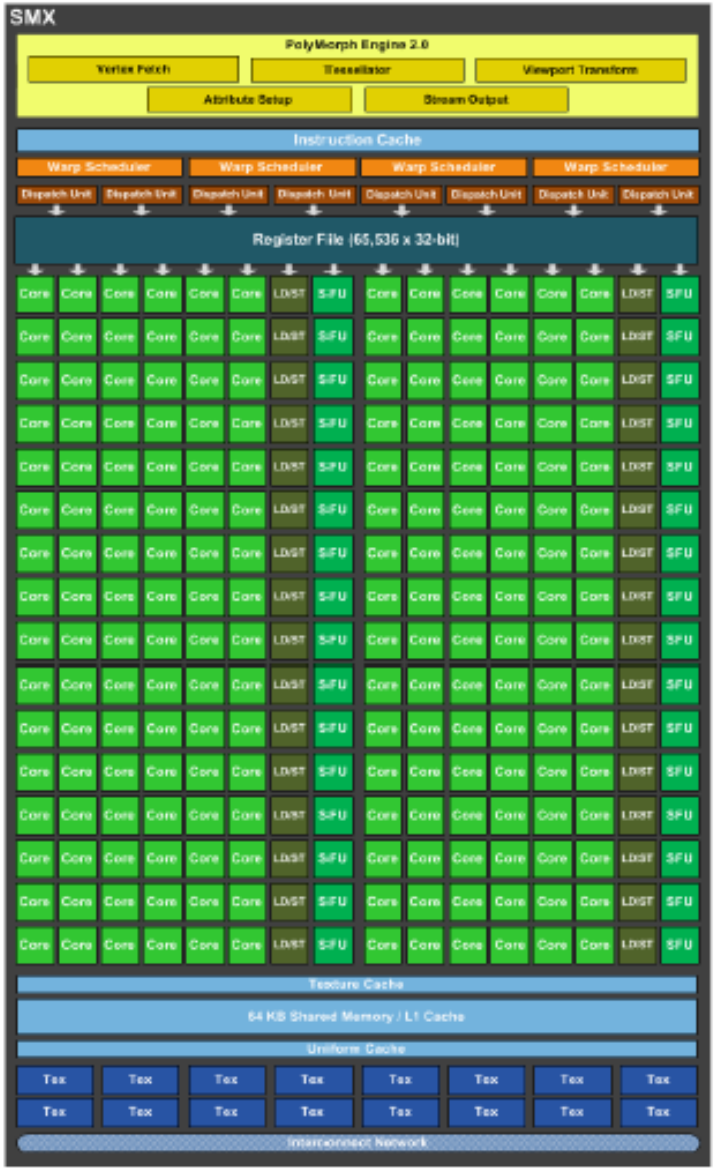
Arquitectura CUDA – Kepler

- **Algunos cambios importantes son:**
  - **Tiene 4 GPCs (Graphics Processing Clusters) compuestos por 2 SMXs. En total son 8 SMXs.**
  - **Utiliza PCI Express 3.0 para la comunicación con el host. Hasta la arquitectura Fermi se usaba PCI Express 2.0.**



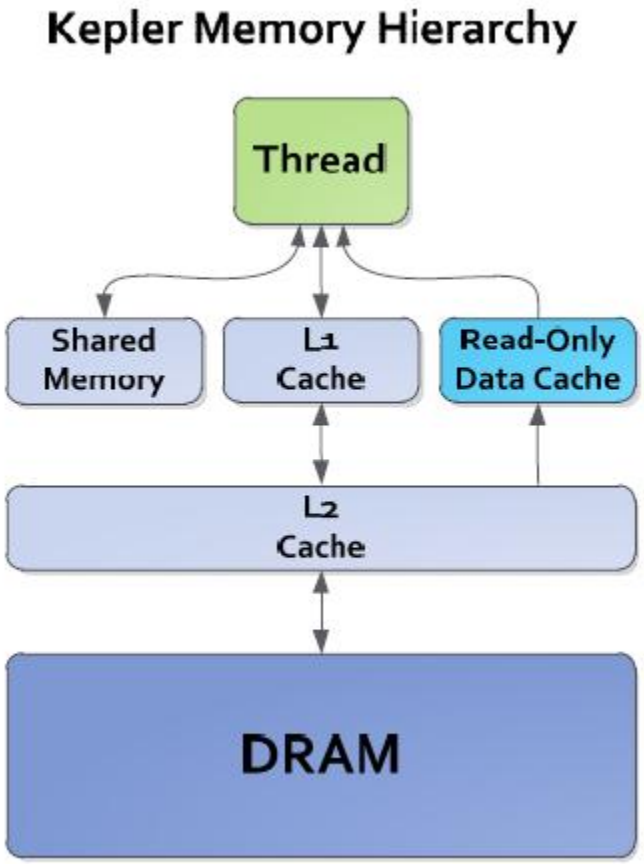
Arquitectura CUDA – Kepler

- **Algunos cambios importantes son:**
  - **Los SMXs están compuestos por 192 CUDA cores. En total son 1536 CUDA cores.**
  - **Tienen 32 SFUs por SMXs. En total son 256 SFUs.**
  - **Tienen 4 Warp Schedulers aunque la cantidad total se mantiene en 32.**



Arquitectura CUDA – Kepler

- **Algunos cambios importantes son:**
  - **Se incorpora un caché de sólo lectura de 48K.**



Arquitectura CUDA – Kepler

	G80	GT200	Fermi	Kepler
Transistores	681 mill	1.4 bill	3.0 bill	3.54 bill
CUDA cores	128	240	512	1536
Core clock	1350 Mhz	1476 Mhz	1544 Mhz	1006 Mhz
Thermal Design Power	155 W	183 W	244 W	195 W
Theorical GFLOPS/s	518	933-1062	1581	3090
GFLOPS/W	3.34	5.80	6.48	15.85
Memory Bandwidth	86.4 GB/sec	159 GB/sec	192.4 GB/sec	192.26 GB/sec

Arquitectura CUDA – Kepler

	Kepler	Maxwell	Pascal	Volta
Transistores	3.54 bill	5.2 bill	7.2 bill	21.1 bill
CUDA cores	1536	2048	2560	5120
Core clock	1006 Mhz	1126 Mhz	1607 Mhz	1200 Mhz
Thermal Design Power	195 W	165 W	180 W	250 W
Theorical GFLOPS/s	3090	4612	8228	13800
GFLOPS/W	15.85	27.95	45.71	55.2
Memory Bandwidth (GB/sec)	192.26	224	320	652.8

- **Maxwell: fines de 2014 (GTX 980)**
- **Pascal: abril-mayo 2016 (GTX 1080)**
- **Volta: diciembre 2017 (GTX Titan V)**

Compute Capabilities

Compute Capabilities

- Para hacer uso óptimo de las GPUs, es necesario conocer diferentes características de la tarjeta.
- Para ello Nvidia utiliza un formato estandarizado para especificar estas características denominado compute capabilities.
- La categorización incluye dos números.
- Los cambios en la primera cifra implican cambios de generación, mientras que en la segunda implica una revisión.
- Las primeras GPUs de CUDA eran compute capability 1.0.
- Las últimas GPUs son:
  - arquitectura Ampere: compute capabilities 8.0 a 8.7
  - arquitectura Ada Lovelace: compute capabilities 8.9
  - arquitectura Hopper: compute capabilities 9.0

Compute Capabilities

Compute Capability Version	Chip de la GPU	Modelos de tarjetas
1.0	G80, G92, GF106	GeForce 8800GTX, 9800GT, Tesla C/D/S870, GT420, GT430, GT440
2.0	GF100, GF110	GeForce GTX 465, 470, 480, 570, 580, 590, Tesla C2050, C2070, S/M2050/70
3.7	GK210	Tesla K80
5.2	GM200, GM204, GM206	GeForce GTX Titan X, 980 y 980 Ti, Tesla M40
6.1	GP100, GP102, GP104, GP106, GP107,GP108	Nvidia Titan X, GeForce GTX 1080 y 1080 Ti, Tesla P40, Tesla P100
7.0	GV100	Nvidia Titan V, Tesla V100

- **Han existido las compute capabilities: 1.0, 1.1, 1.2, 1.3, 2.0, 2.1, 3.0, 3.2, 3.5, 3.7, 5.0, 5.2, 5.3, 6.0, 6.1, 6.2, 7.0, 7.2, 7.5, 8.0, 8.6, 8.7, 8.9 y 9.0.**

Compute Capabilities

	1.0	2.x	3.7	5.2	6.1	7.0
Tamaño de warp	32	32	32	32	32	32
Warps residentes por MP	24	48	64	64	64	64
Hilos residentes por MP	768	1536	2048	2048	2048	2048
Registros por MP	8 K	32 K	128 K	64 K	64 K	64 K
Dimensión máxima del grid	2	3	3	3	3	3
Máximo de instrucciones por kernel	2 millones	512 millones				