Programación de GPUs con CUDA

Programación de Arquitecturas Multinúcleo
Grado en Ingeniería Informática

Curso 2023/24

Contenidos

1. Introducción

- Procesadores de propósito general
  - En las últimas décadas del siglo XX los procesadores comerciales aumentaban su rendimiento año tras año
  - En los últimos años han surgido algunas dificultades:
    - El consumo de energía limita la frecuencia de reloj
    - El paralelismo de las aplicaciones secuenciales que se puede explotar de manera transparente es limitado
- Consecuencia: coprocesadores/aceleradores acompañando a la CPU:
  - Many Integrated Core (MIC):
    - Coprocesador Intel Xeon Phi: hasta 61 núcleos, 244 threads (abandonado en 2018)
  - GPUs:
    - nVIDIA principalmente
    - AMD
    - GPUs Intel Max
  - Aceleradores Cerebras (IA)

1. Introducción

- CUESTIONES CLAVE:
  - ¿Qué diferencias hay entre un procesador multinúcleo de propósito general y una GPU?
  - ¿Qué ofrecen las GPUs que las hace atractivas para aprovecharlas para realizar otras tareas?
  - ¿Qué características de las GPUs condicionan su utilización para realizar esas tareas?

1. Introducción

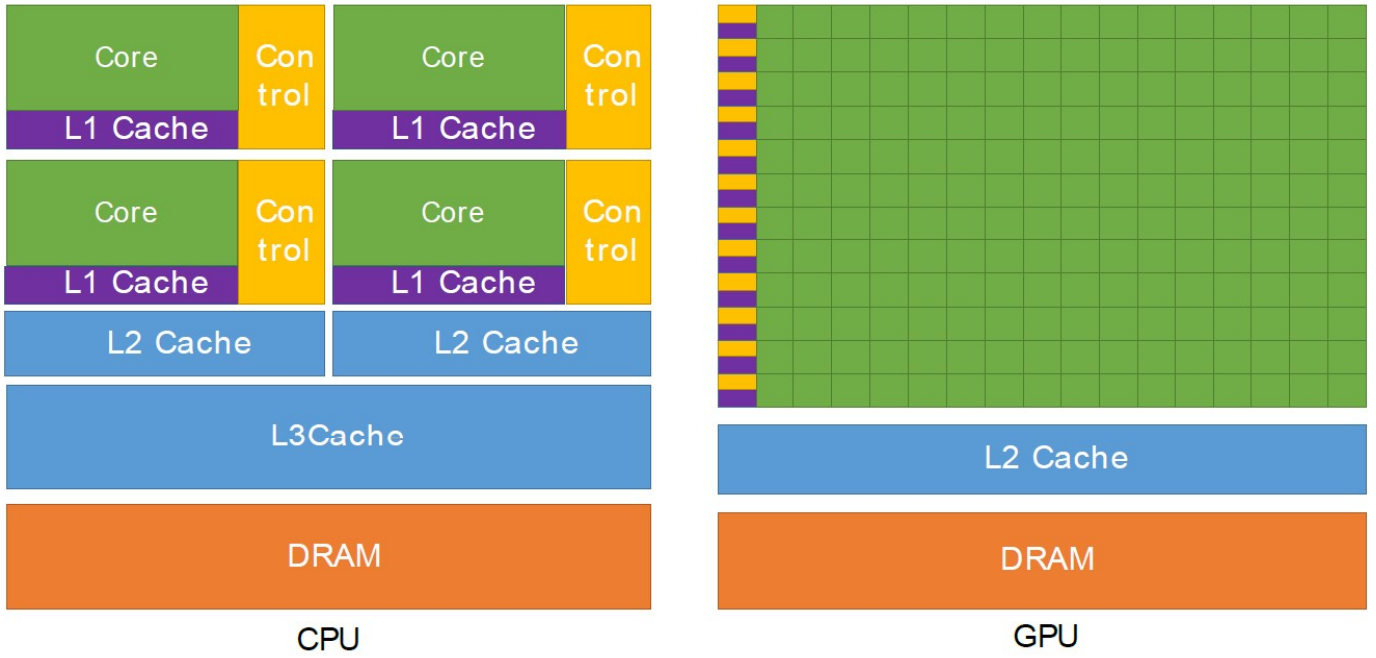
- ***Graphics Processing Units* (GPUs)**
  - Las GPUs liberan a la CPU de realizar **tareas concretas** de procesamiento gráfico de manera repetitiva
  - Presentes en **cualquier equipo** de sobremesa o servidor integradas en placa o como tarjetas externas
    - ISA VESA PCI AGP 1/2/4/8x PCIe x1/4/8/16
  - El amplio mercado de los **vídeojuegos** ha propiciado su consolidación, rápida evolución y precios competitivos
  - Las **GPUs** actuales también son **procesadores multinúcleo** porque el procesamiento gráfico es inherentemente paralelo
    - La necesidad de ejecutar múltiples operaciones en punto flotante para procesar cada imagen…
    - …se satisface mediante muchos *threads* capaces de ejecutarse en paralelo

1. Introducción

- GPUs ofrecen mayor rendimiento pico que las CPUs
  - CPUs diseñadas para optimizar la ejecución de aplicaciones de propósito general
    - Lógica de control flujo muy sofisticada
    - Memorias caché multinivel
    - # núcleos: desde 4 hasta 64 (Intel y AMD)
  - GPUs diseñadas para optimizar la ejecución de tareas de procesamiento gráfico (computación intensiva)
    - Lógica de control de flujo simple y memorias caché pequeñas
      - Mismo programa para cada dato → lógica de control sencilla
      - Ejecución intensiva en cálculo sobre muchos datos → latencias de acceso a memoria se ocultan con cálculos en lugar de usando grandes caches
    - Múltiples unidades funcionales para punto flotante
    - Mayor ancho de banda de acceso a memoria
    - # núcleos: 1000 en adelante...

1. Introducción

Figure 1. The GPU Devotes More Transistors to Data Processing



¿Por qué hay discrepancia en la capacidad de cómputo GPU vs. CPU ?

GPU:

- Especializada en computación altamente paralela (renderizado gráfico)
- Más transistores dedicados a procesamiento. Menos a cachés y control de flujo

1. Introducción

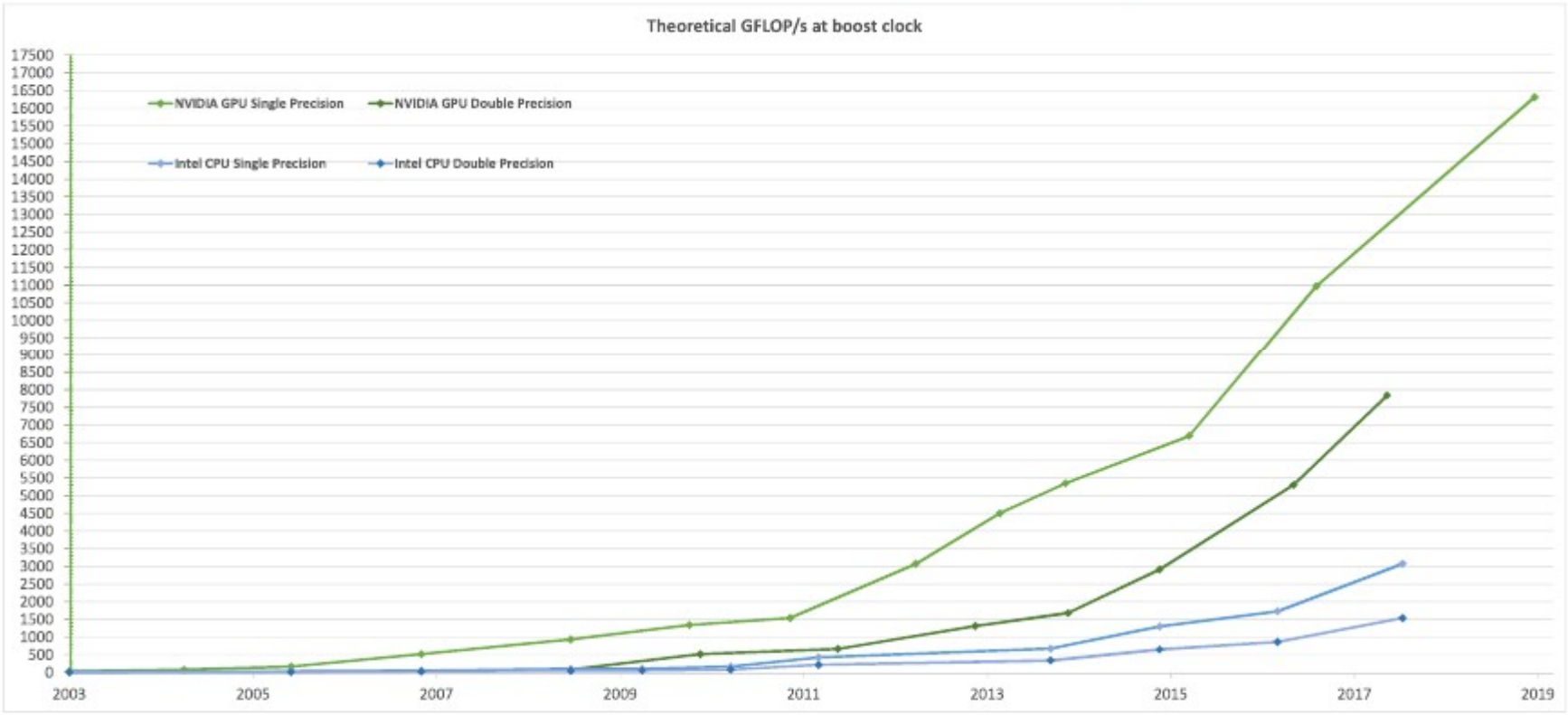


Figure 1 Floating-Point Operations per Second for the CPU and GPU

1. Introducción

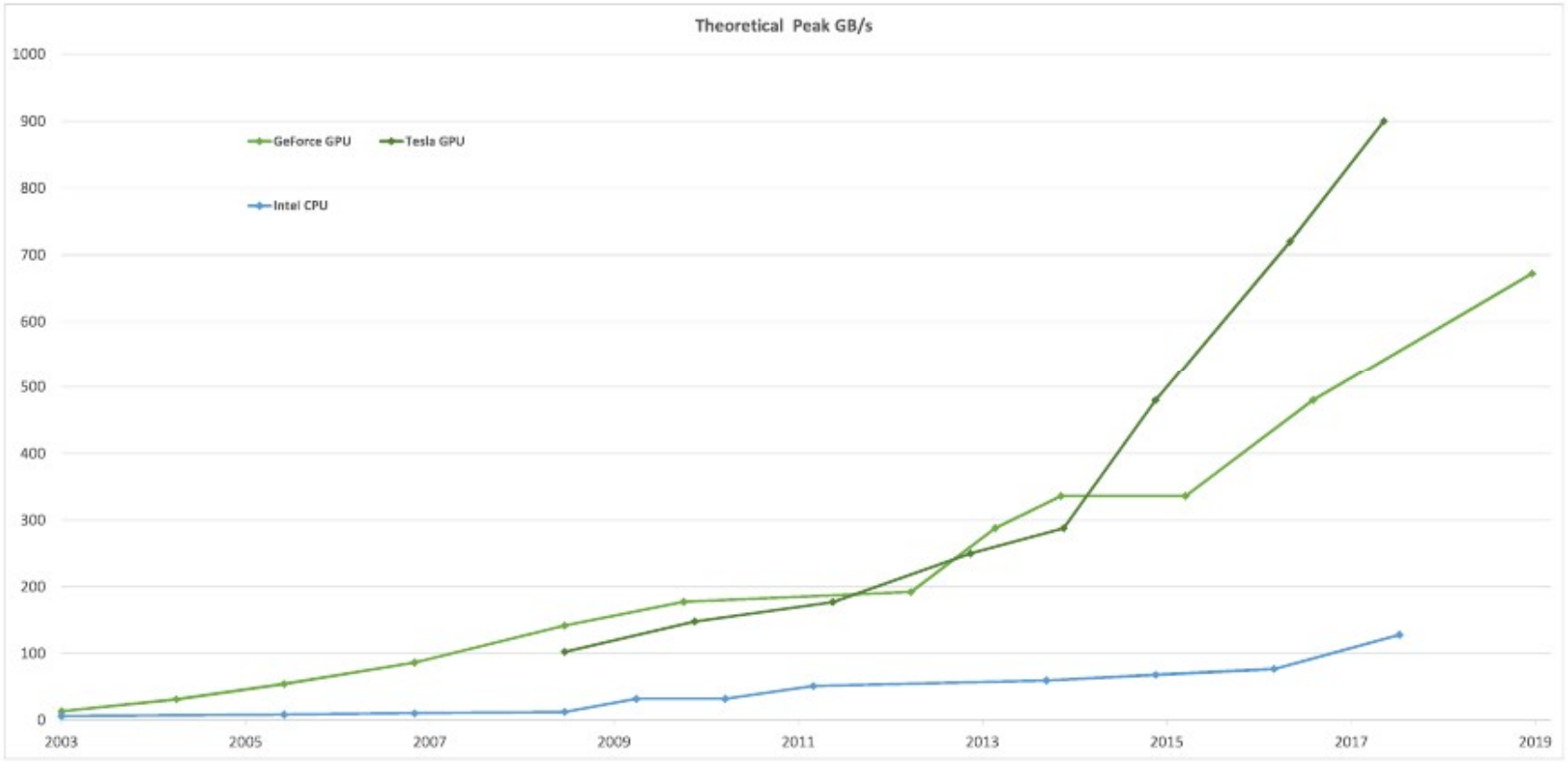
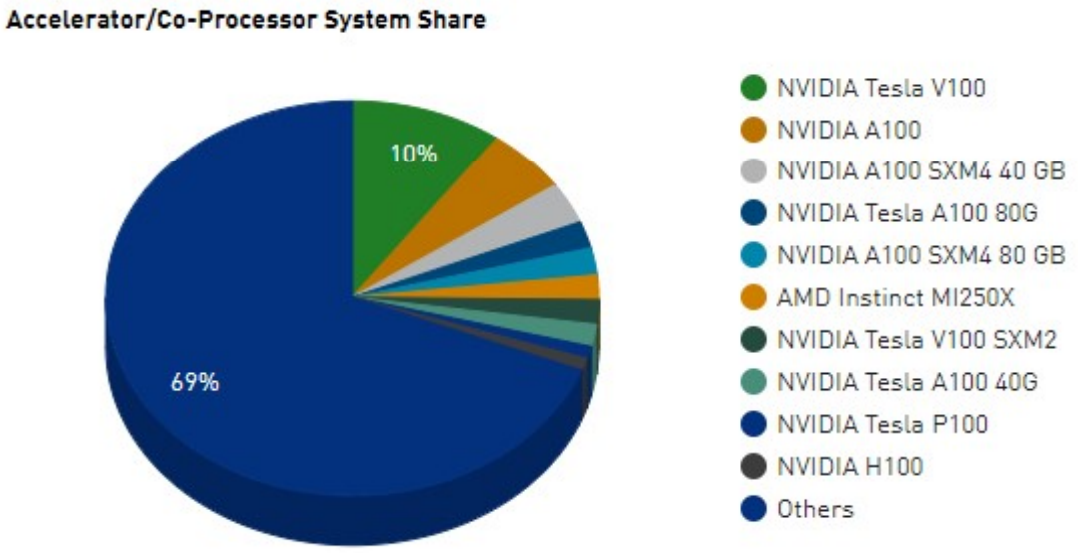


Figure 2 Memory Bandwidth for the CPU and GPU

1. Introducción

- TOP500: noviembre 2023



1. Introducción

- Esta *ventaja* ha despertado el interés por explorar el uso de GPUs para acelerar aplicaciones de propósito general

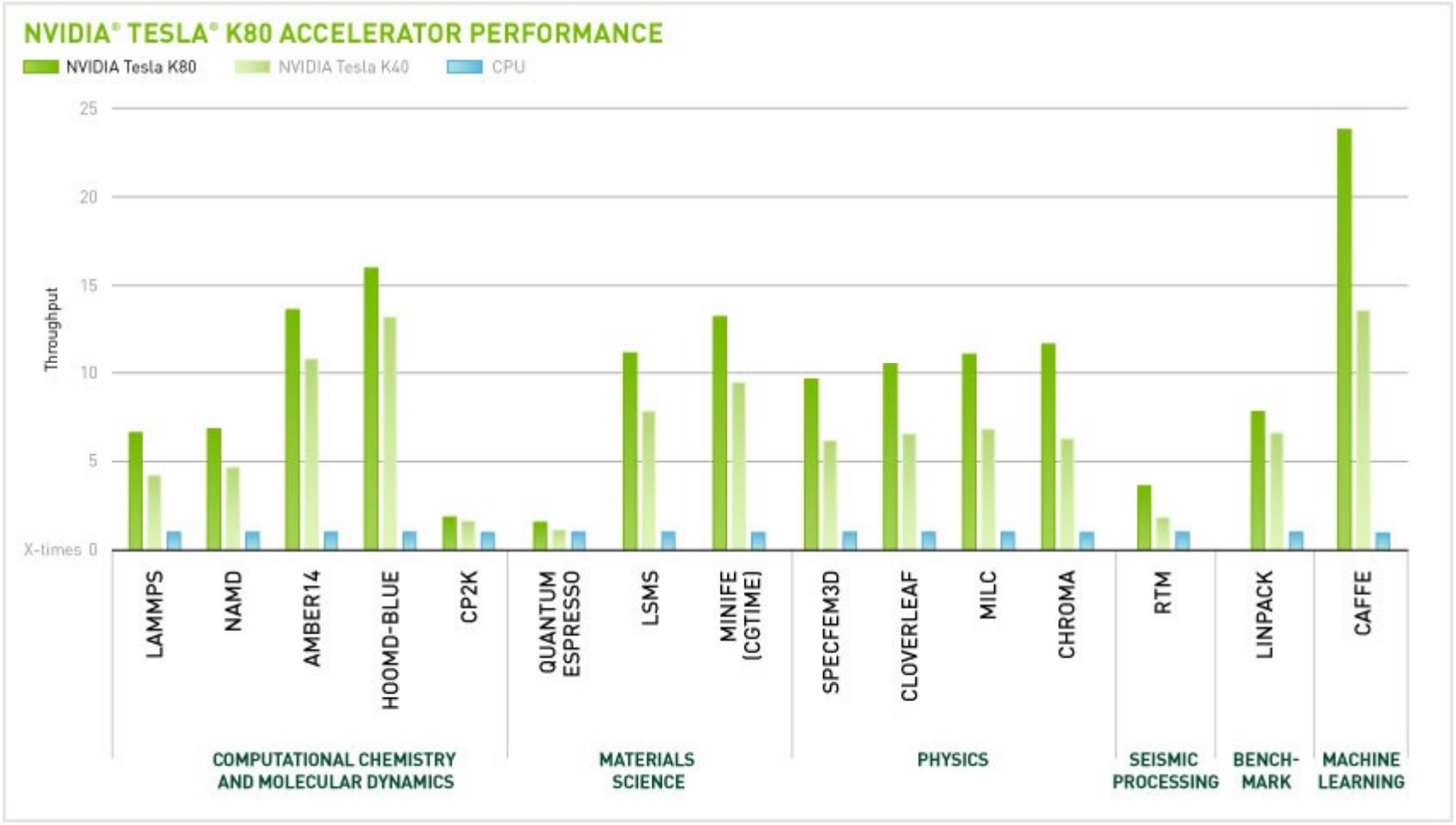
- ***General-Purpose computation on Graphics Processing Units* (GPGPU)**
  - **Programación mediante APIs gráficas (Direct3D,OpenGL)**
    - Modificación de la aplicación para expresarla en función de un conjunto de llamadas a la API gráfica disponible
      - » Tarea ardua y compleja que requiere conocimiento detallado tanto de la arquitectura de la GPU como de la aplicación
      - » La API limita las aplicaciones que pueden adaptarse
  - **CUDA** proporciona un modelo de programación independiente de las APIs gráficas mucho más general y flexible
    - Las aplicaciones también DEBEN paralelizarse (NVIDIA HPC SDK)

1. Introducción

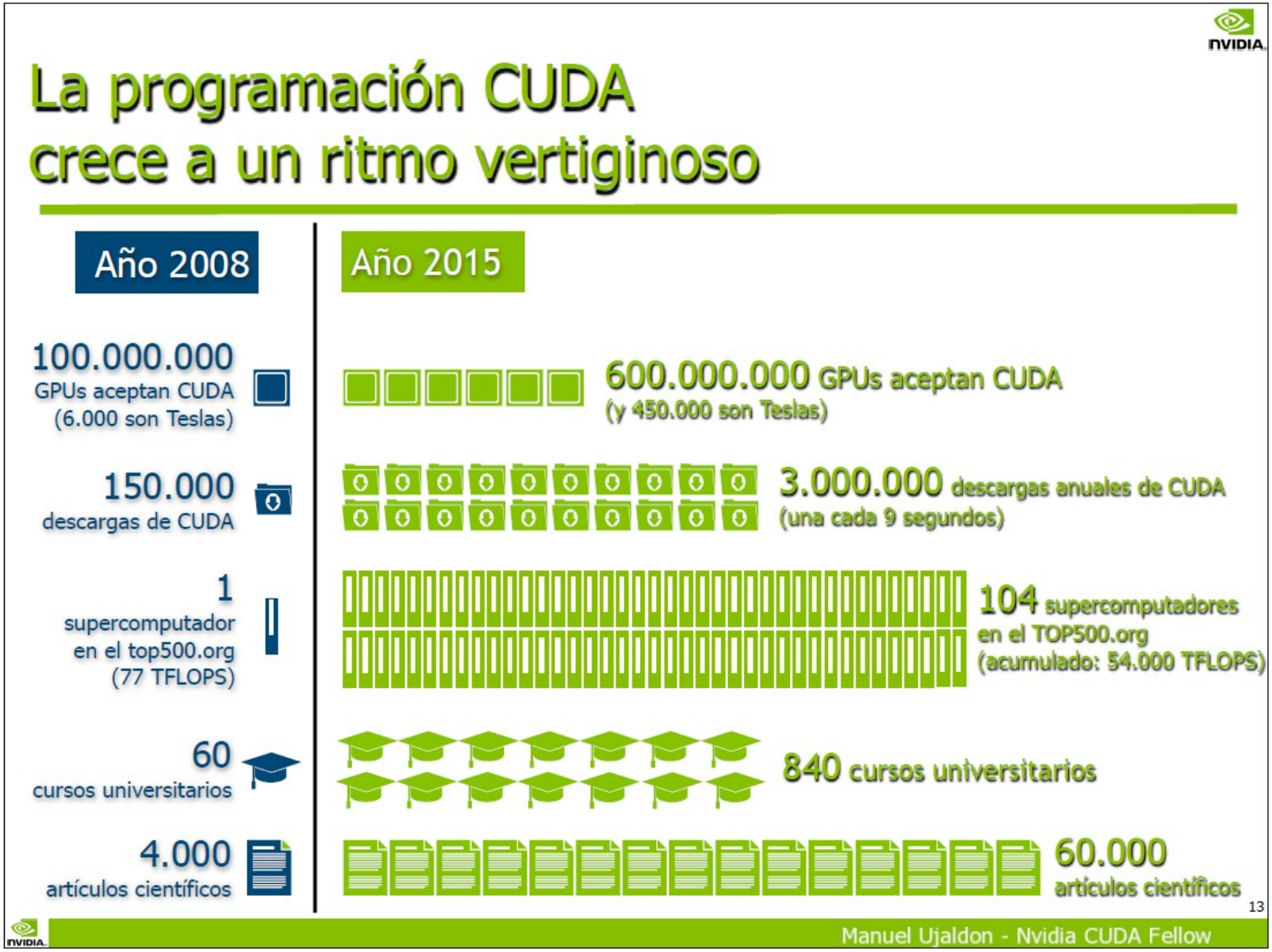
- GPUs (CUDA) ofrecen mayor rendimiento efectivo que las CPUs en aplicaciones de diversos campos:
  - Cálculo matricial denso
  - Resolución ecuaciones polinomiales
  - …
- Desafortunadamente no todas las aplicaciones son susceptibles de ser paralelizadas con éxito en GPUs
- Pero, cada vez hay más campos de aplicación:
  - Bioinformática, Inteligencia artificial, Bases de datos, Robótica,…

https://www.nvidia.com/es-es/industries/supercomputing/

1. Introducción

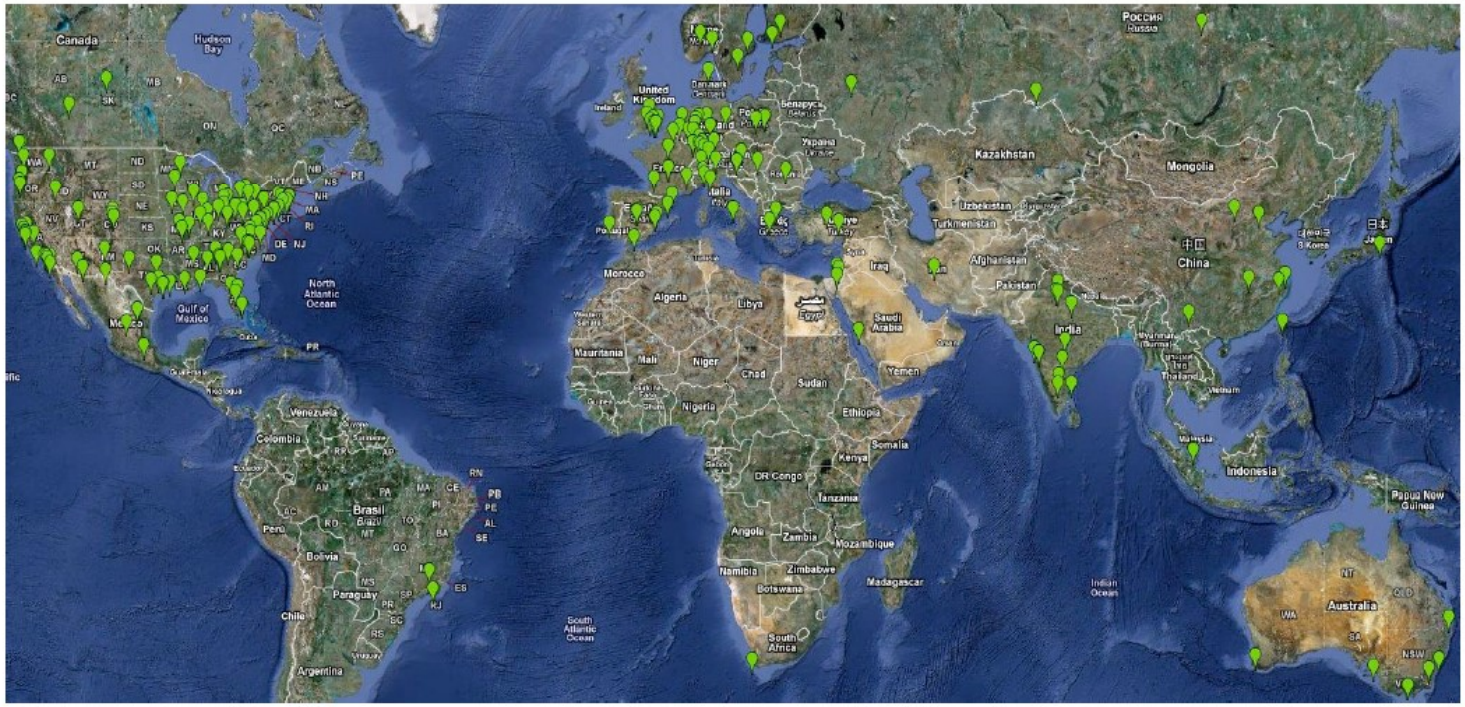


1. Introducción



1. Introducción

Distribución mundial de las 840 universidades que imparten cursos de CUDA



14

Manuel Ujaldon - Nvidia CUDA Fellow

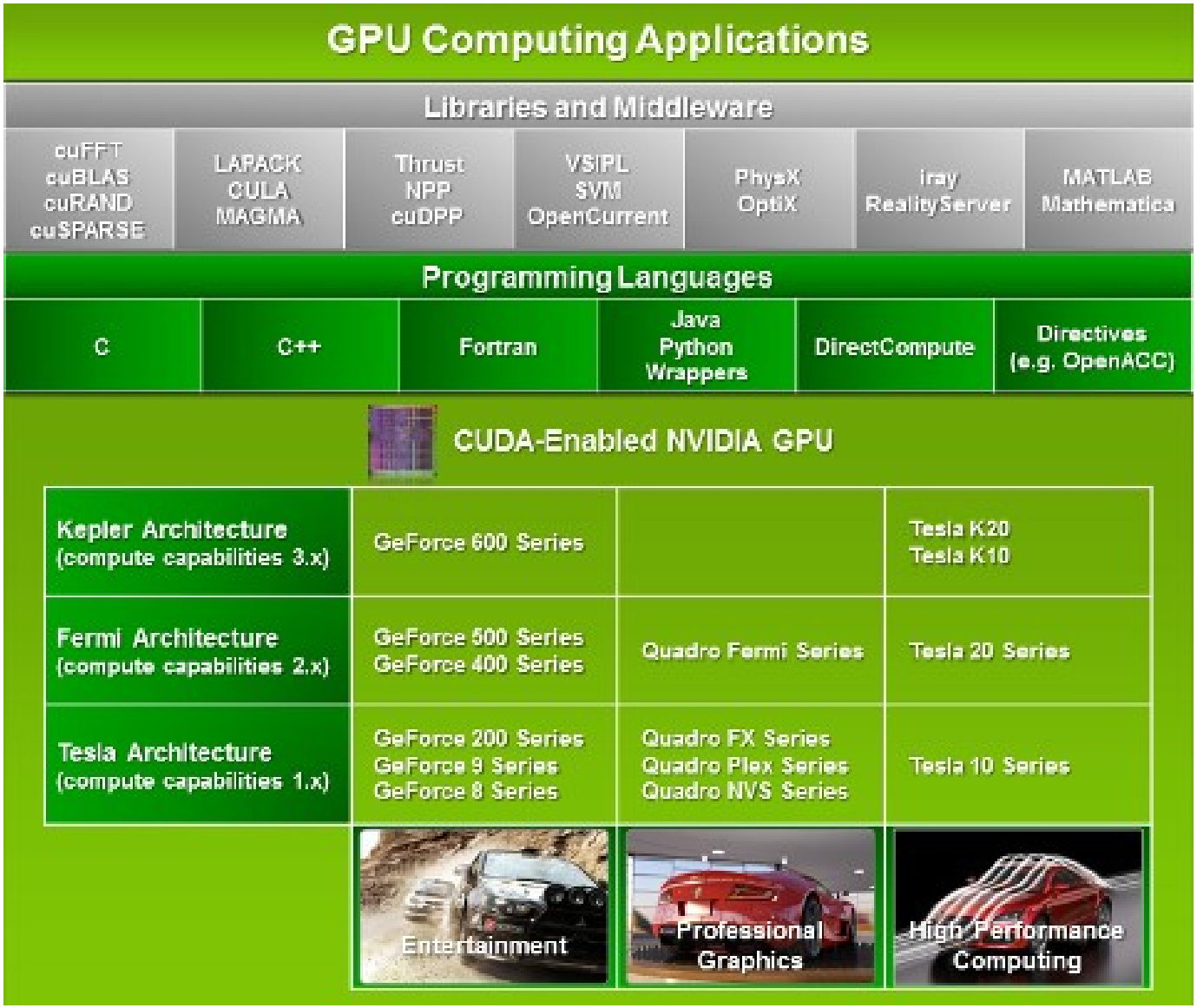
1. Introducción

- En NOV'06 NVIDIA introduce la arquitectura unificada: GeForce 8800
- CUDA: Compute Unified Device Architecture
- GPUs de NVIDIA compatibles con CUDA:
  - Tesla -> Compute Capability 1.x
  - Fermi -> Compute Capability 2.x
  - Kepler -> Compute Capability 3.x
  - Maxwell -> Compute Capability 5.x
  - Pascal -> Compute Capability 6.x
  - Volta -> Compute Capability 7.x
  - Turing -> Compute Capability 7.x
  - Ampere -> Compute Capability 8.x
  - Hopper -> Compute Capability 9.x
- Diferencias:
  - Interfaz (ancho de banda) y memoria integrada (MB)
  - *Compute Capability* (1.x , 2.x, …)
    - Recursos: # núcleos (# SMs y # SPs por SM), …
    - Funcionalidad: soporte IEEE 754 DP, …

    Más información:

    CUDA C Programming Guide . Appendix I. Compute Capabilities

1. Introducción



1. Introducción

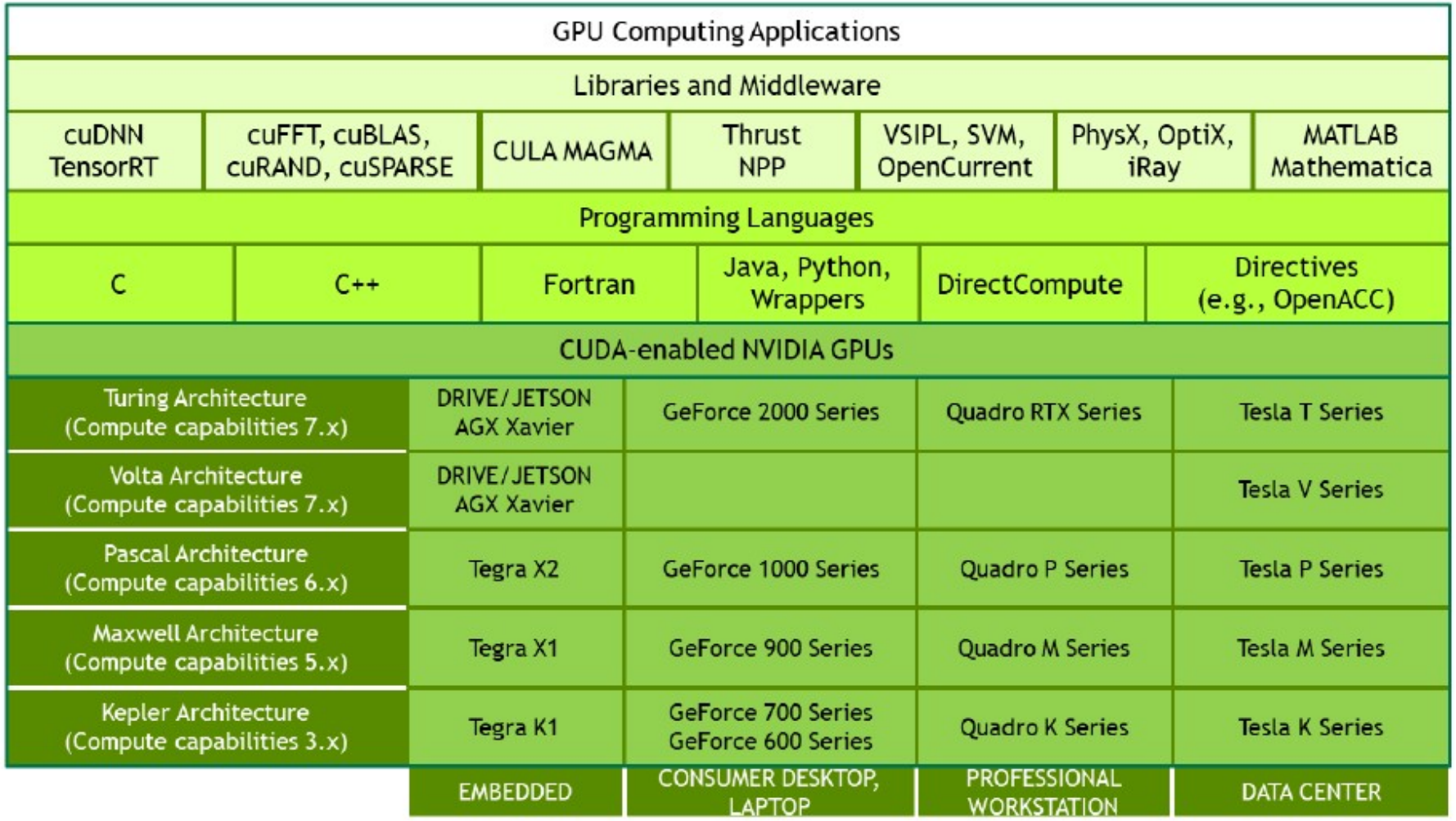


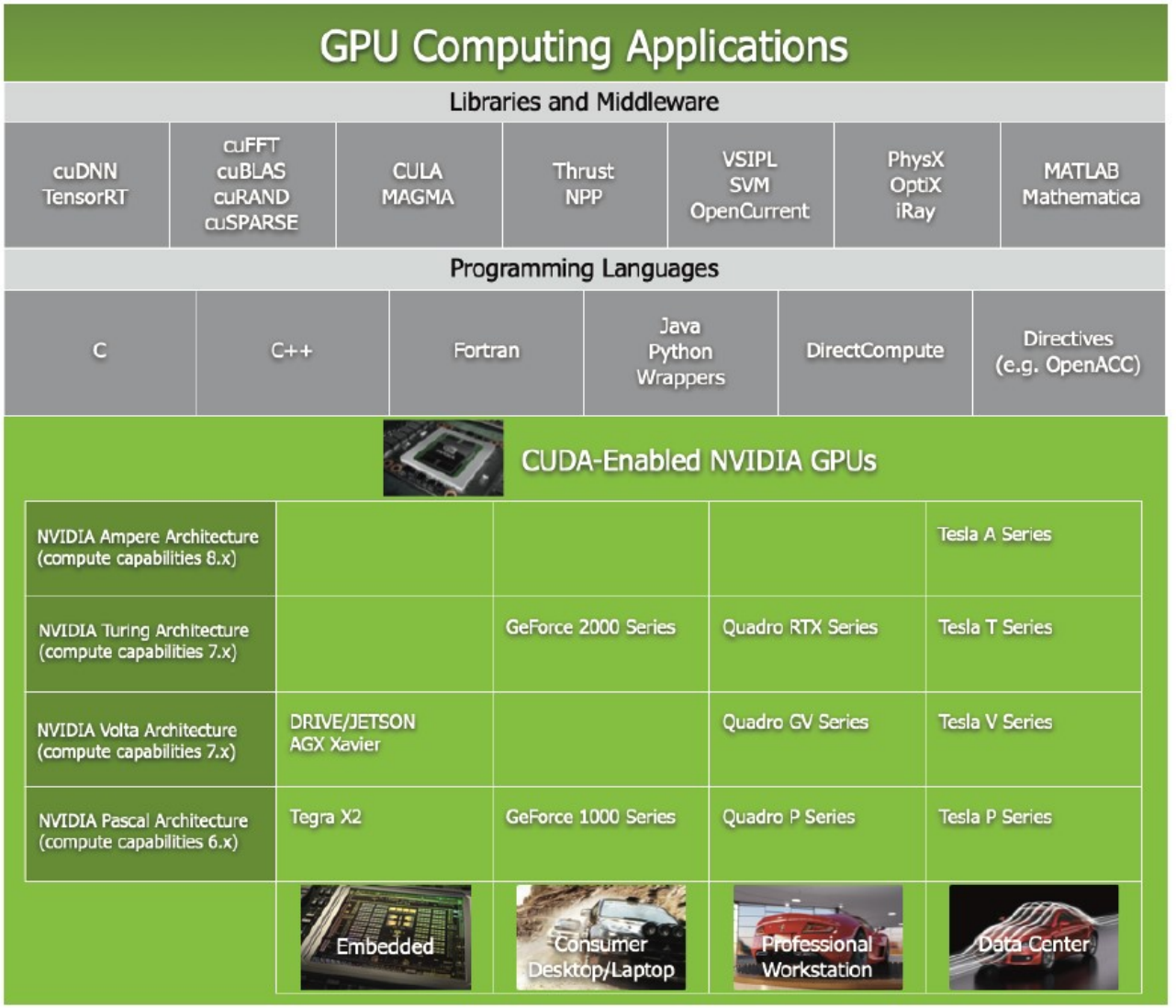
Figure 4 GPU Computing Applications

CUDA is designed to support various languages and application programming interfaces.

1. Introducción

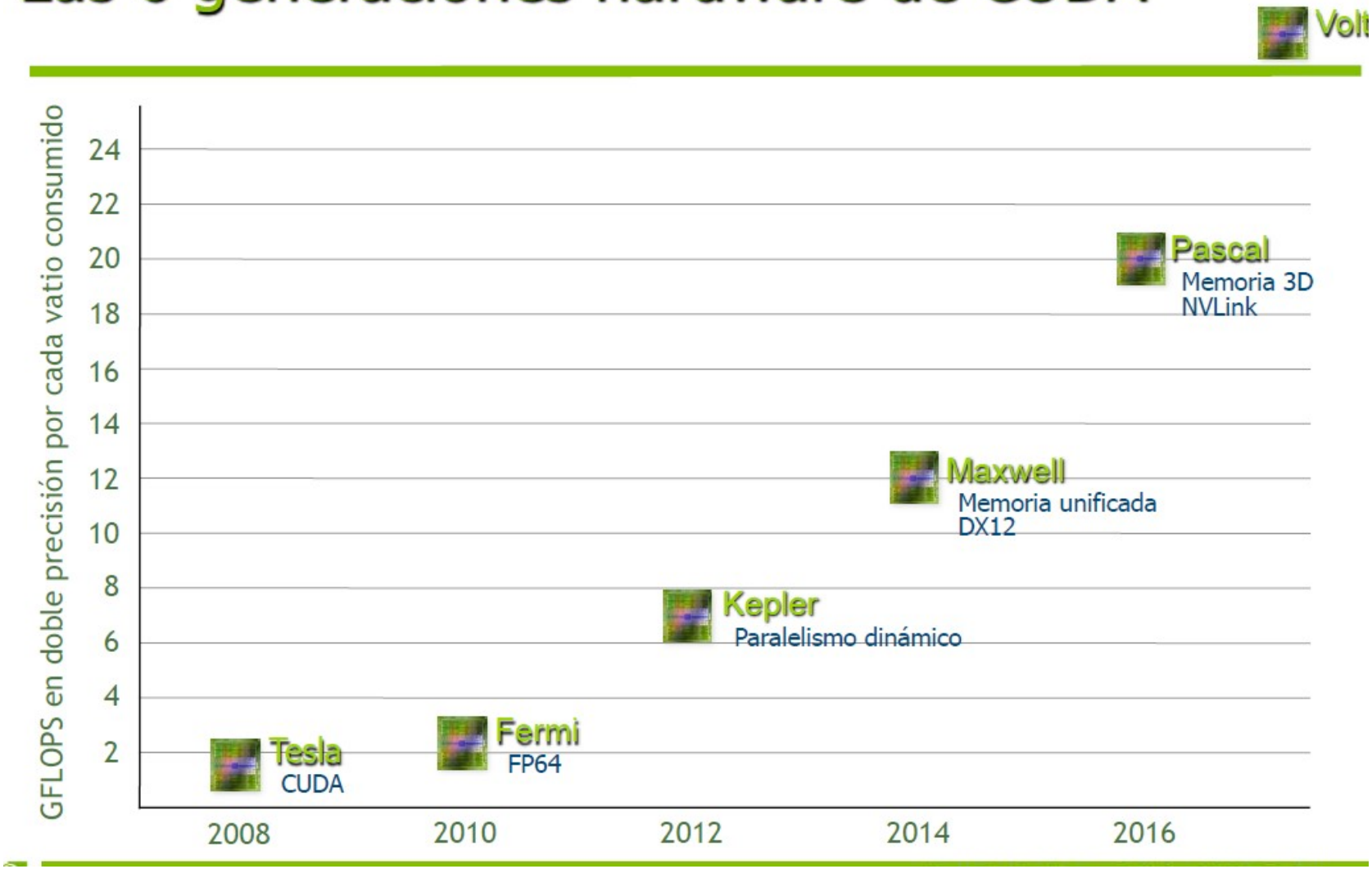
Figure 2. GPU Computing Applications

CUDA is designed to support various languages and application programming interfaces.



1. Introducción

Las 6 generaciones hardware de CUDA



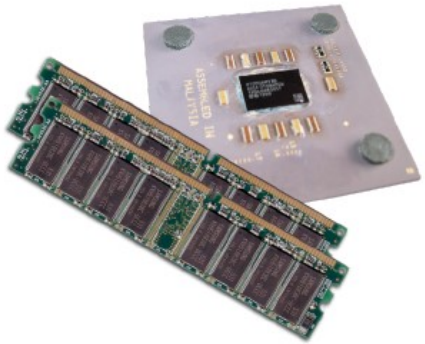
1. Introducción

- ***Compute Unified Device Architecture* (CUDA)**
  - Arquitectura hardware y software
    - Uso de GPU, construida a partir de la replicación de un bloque constructivo básico, como acelerador con memoria integrada
    - Estructura jerárquica de *threads* mapeada sobre el hardware
  - Modelo de memoria: Gestión de memoria explícita
  - Modelo de ejecución: Creación, planificación y ejecución transparente de miles de *threads* de manera concurrente
  - Modelo de programación: Extensiones del lenguaje C/C++ junto con CUDA *Runtime* API

1. Introducción

Computación heterogénea (1/4)

- Terminología:
  - Host (el anfitrión): La CPU y la memoria de la placa base [DDR3].
  - Device (el dispositivo): La tarjeta gráfica [GPU + memoria de vídeo]:
    - GPU: Nvidia GeForce/Tesla.
    - Memoria de vídeo: GDDR5 en 2015.



Host

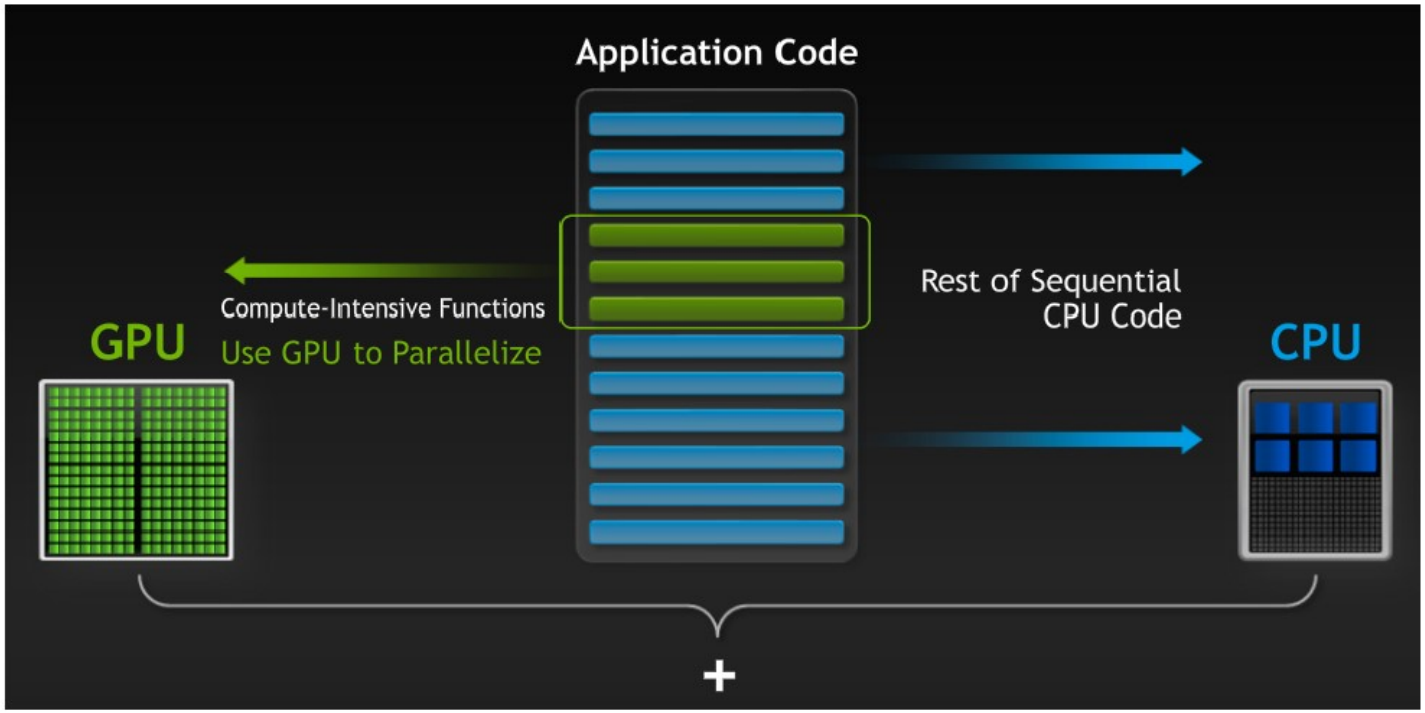
Device

24

Manuel Ujaldon - Nvidia CUDA Fellow

1. Introducción

Computación heterogénea (3/4)



- El código reescrito en CUDA puede ser inferior al 5%, pero consumir más del 50% del tiempo si no migra a la GPU.

26

Manuel Ujaldon - Nvidia CUDA Fellow

1. Introducción

- Todo lo necesario para instalar CUDA, así como manuales y programas de ejemplo se puede encontrar en:
  - http://developer.nvidia.com/cuda-toolkit
  - http://developer.nvidia.com/cuda-zone
  - http://docs.nvidia.com/cuda/

Contenidos

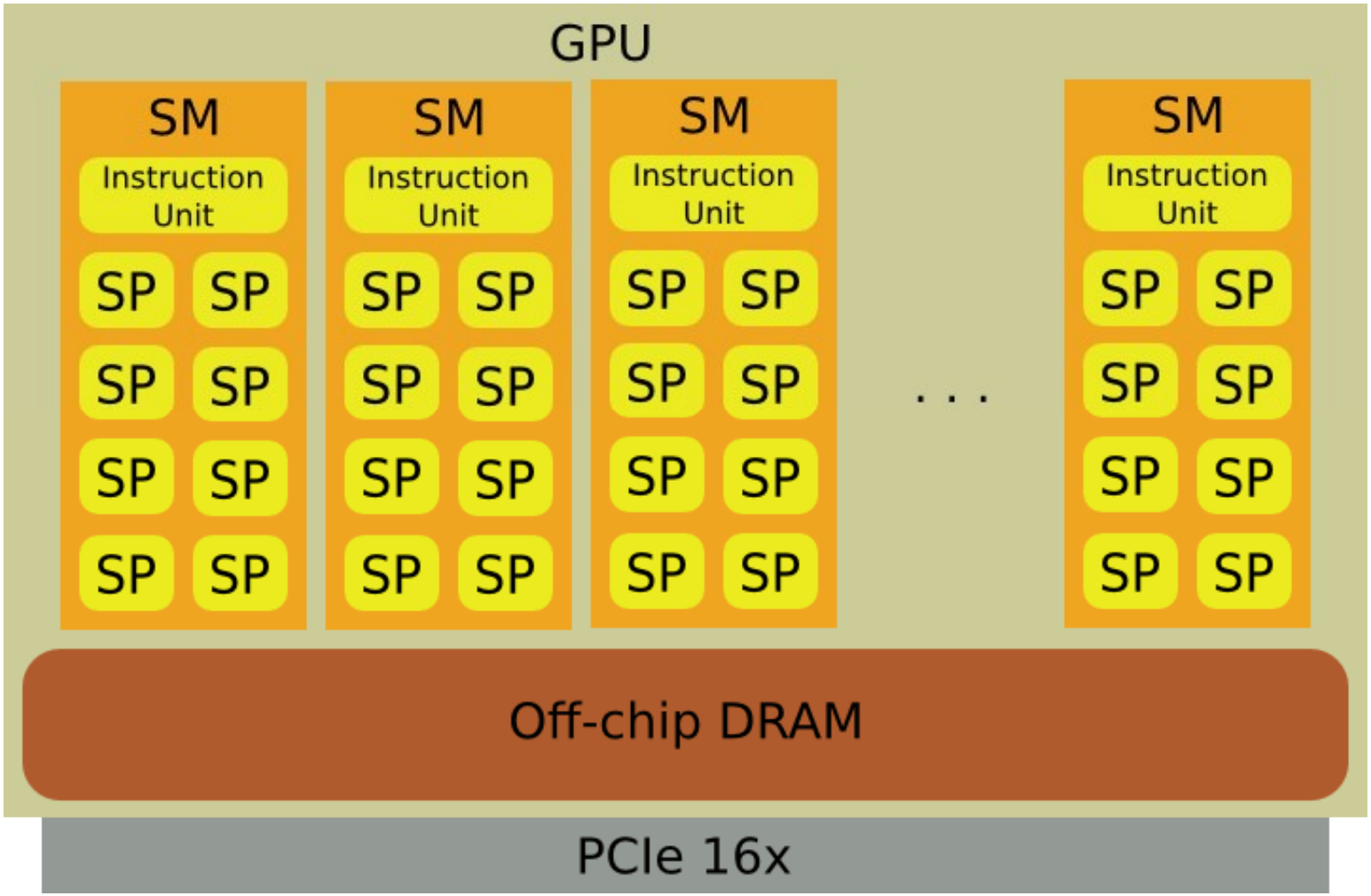
Contenidos

Contenidos

2.1. Arquitectura hardware y software

- **GPU** => conjunto de *N Streaming Multiprocessors* (**SMs**)
- **SM** => conjunto de *M Streaming Processors* (**SPs**)
- **Conjunto de SPs de un SM:**
  - Realizan operaciones escalares sobre enteros de 32 bits, reales SP y reales DP (a partir de 2.x) (compatible con IEEE 754)
  - Ejecutan *threads* independientes pero…
  - …todos deberían ejecutar la instrucción leída por la *Instruction Unit* (IU) para optimizar el rendimiento
    - *Single Instruction Multiple Thread* (SIMT)
      - » Explotación paralelismo de datos y, en menor grado, de tareas
- **Los *threads* gestionados por el hardware en cada SM**
  - Creación/cambios de contexto con coste despreciable
  - Se libera al programador de realizar estas tareas
  - Ejecución de tantos *threads* como sea posible

2.1. Arquitectura hardware y software



2.1. Arquitectura hardware y software

- Las partes del código secuencial paralelizadas para ser ejecutadas por la GPU se denominan ***kernels***

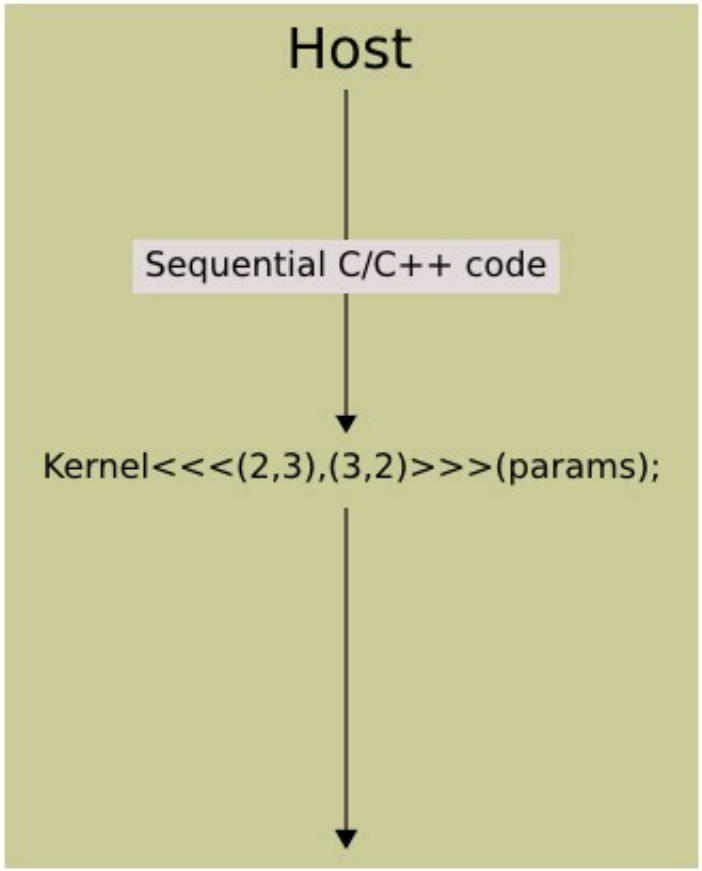
- Un *kernel* descompone un problema en un conjunto de subproblemas *independientes* y lo mapea sobre un *grid*

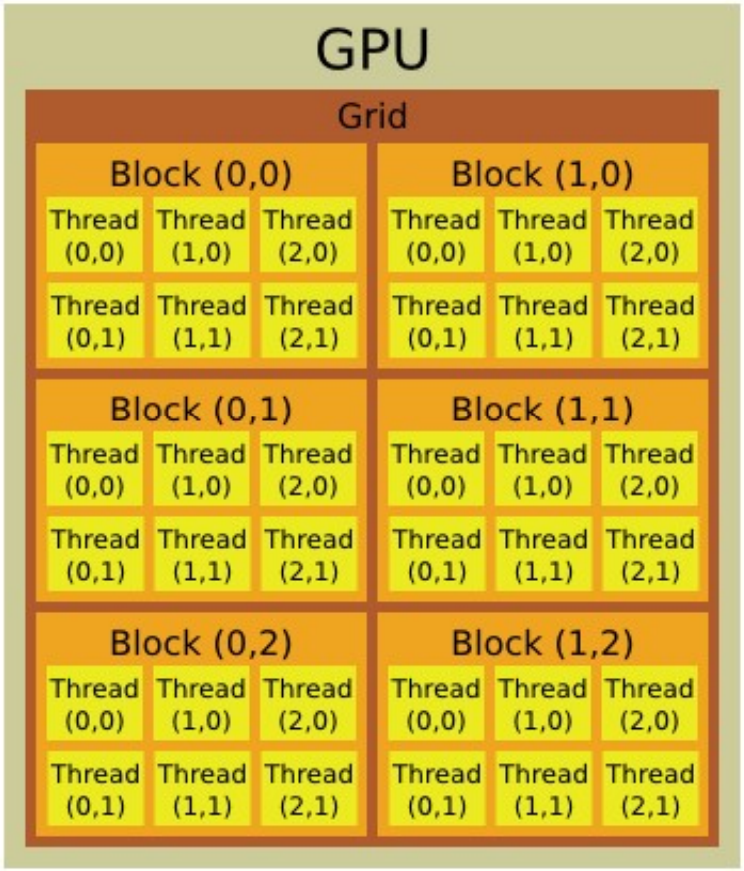
  - ***Grid***: vector 1D, 2D ó 3D de *thread blocks*
    - Cada *thread block* tiene su **BID (X,Y,Z)** dentro del *grid*

  - ***Thread blocks***: vector 1D, 2D ó 3D de *threads*
    - Cada *thread* tiene su **TID (X,Y,Z)** dentro de su *thread block*

- Los *threads* utilizan su BID y su TID para determinar el trabajo que tienen que realizar
  - *Single Program Multiple Data* (SPMD)

2.1. Arquitectura hardware y software





2.1. Arquitectura hardware y software

- Ejemplo: **y=α·x+y,**
- α : escalar
- **x y** : vectores

```
/* Llamada código secuencial */
saxpy_serial(n, 2.0, x, y);
```

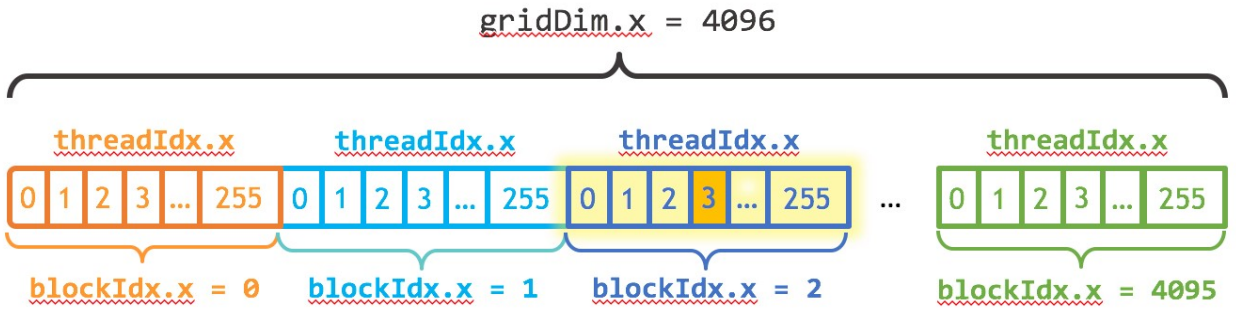
```
void saxpy_serial(int n, , float alpha, float *x , float *y)
{
   for(int i=0; i<n; i++)
     y[i] = alpha*x[i] + y[i];
}
```

2.1. Arquitectura hardware y software

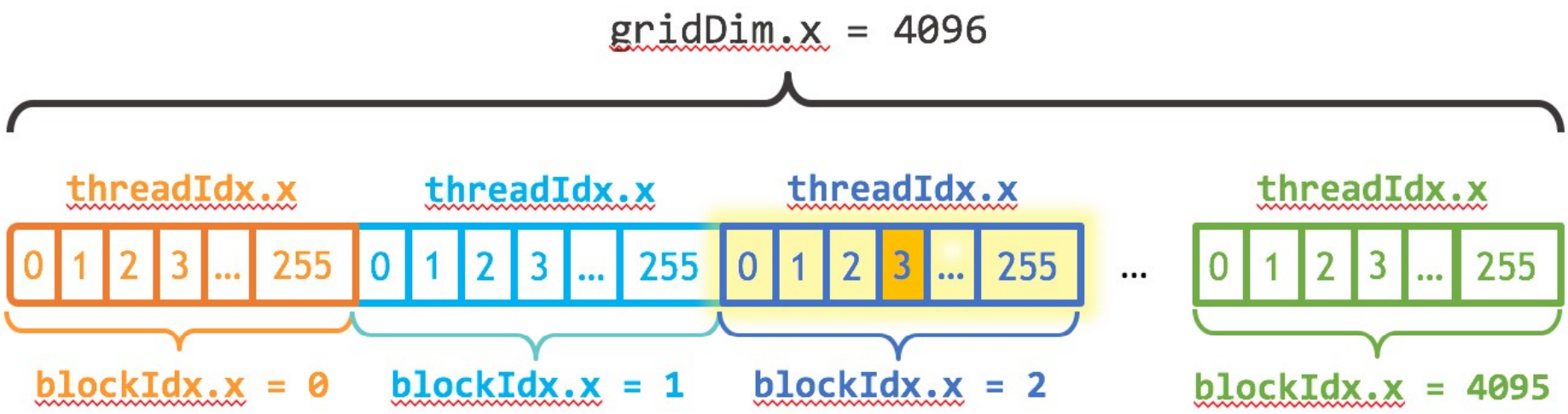
- Ejemplo: **y=α·x+y**
- **/* Llamada código paralelo desde código CPU */**

```
int nblocks = (n + 255)/256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

```
__global__ /* Código GPU */
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
   int index = blockIdx.x * blockDim.x + threadIdx.x;
   if (index < n) y[index] = alpha*x[index] + y[index];
}
```



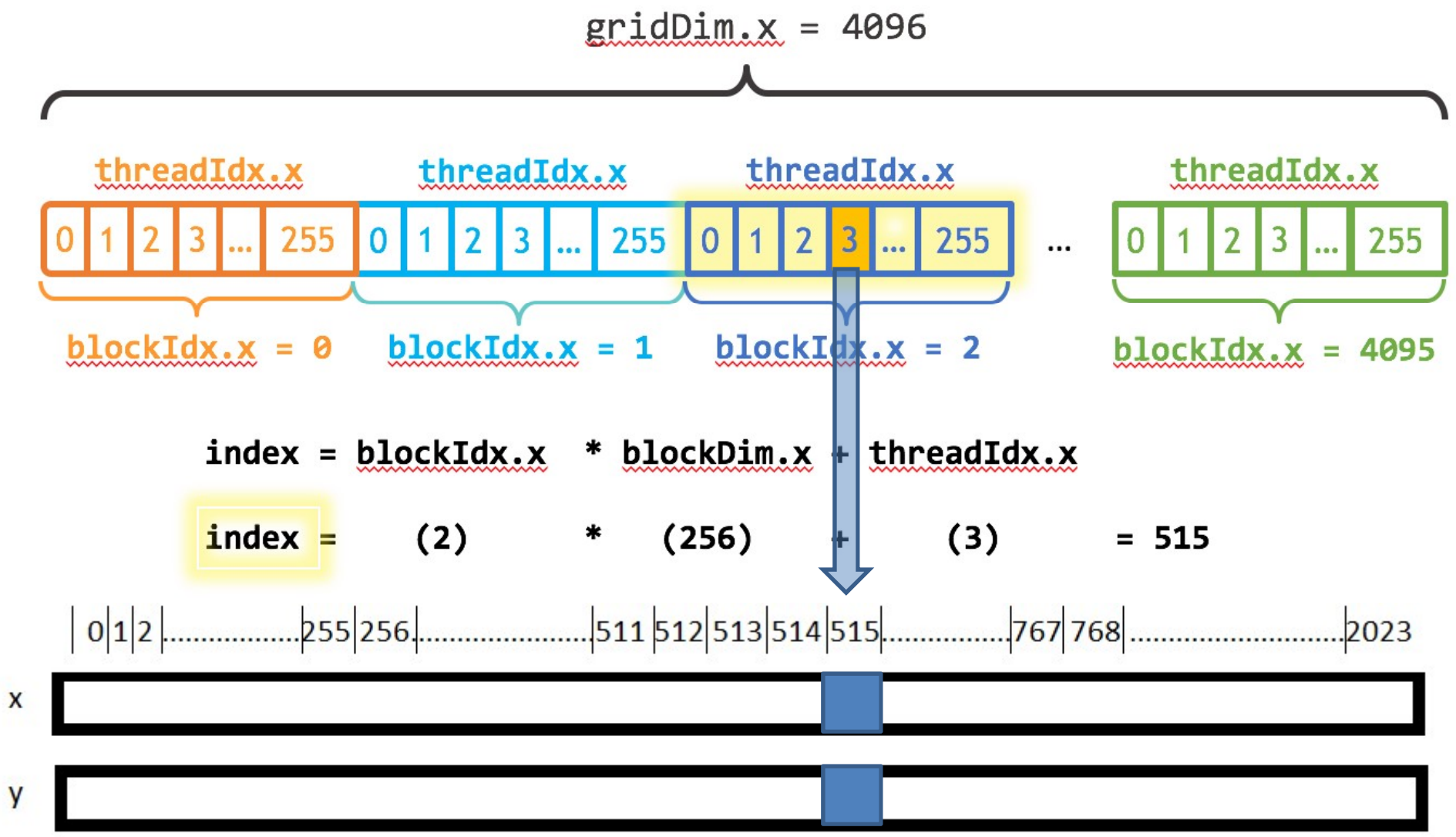
2.1. Arquitectura hardware y software




```
index = blockIdx.x  * blockDim.x + threadIdx.x

index =    (2)      *   (256)    +     (3)      = 515
```

2.1. Arquitectura hardware y software



Contenidos

2.2. Modelo de memoria

- **Banco de registros (*register file*):**
  - Repartidos entre todos los *thread blocks* en ejecución
  - Tiempo de acceso muy pequeño

- **Memoria compartida (*shared memory*):**
  - Repartida entre todos los *thread blocks* en ejecución
  - Compartida por todos los *threads* de cada *thread block*
    - Almacenamiento de datos temporales a modo de caché
  - Tiempo de acceso similar a los registros

2.2. Modelo de memoria

- **Memoria global (*global memory*):**
  - Compartida por todos los *thread blocks*
  - Tiempo de acceso elevado (cientos de ciclos)

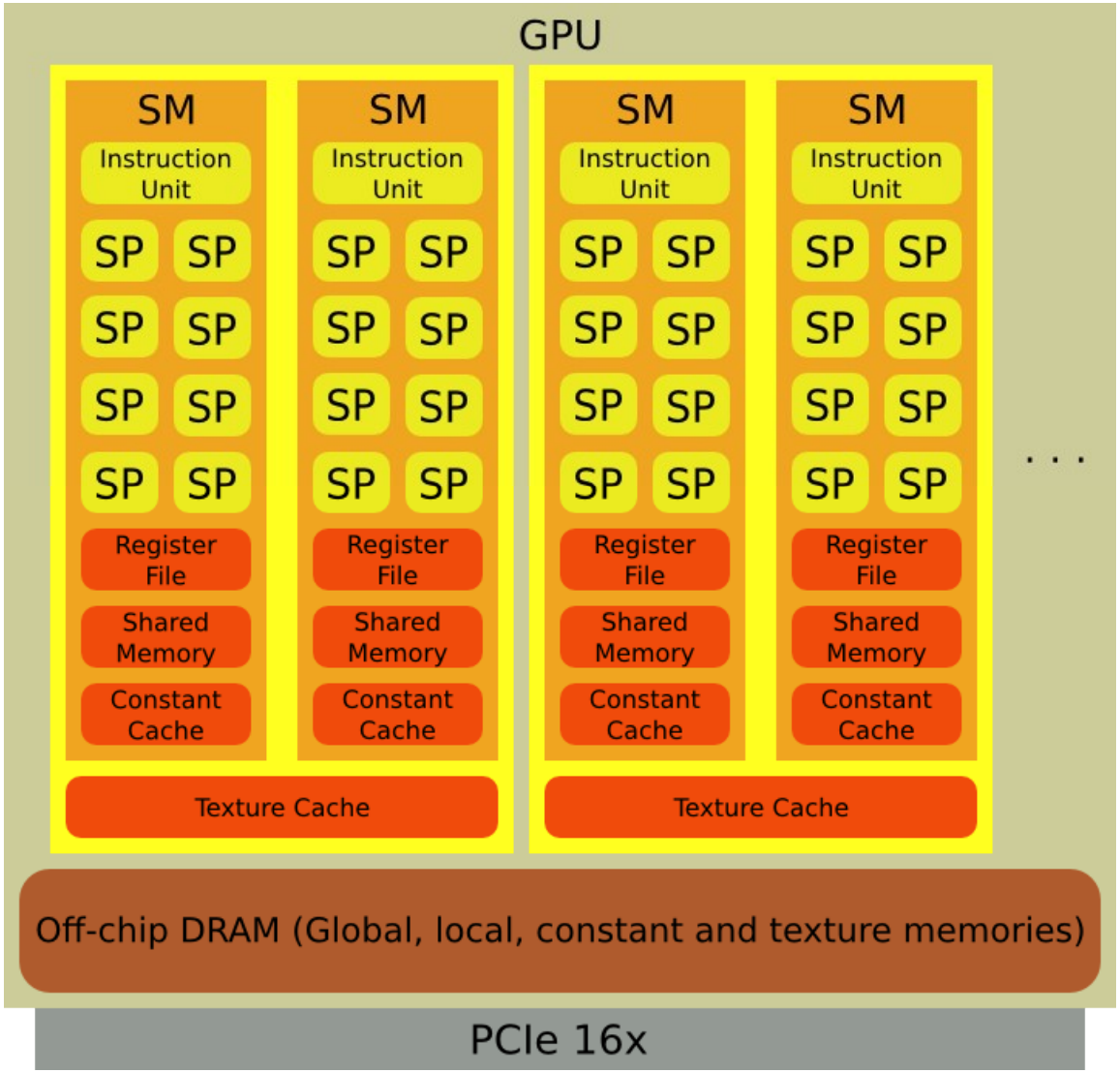
- **Memoria local (*local memory*)**
  - Memoria privada de cada *thread* para la pila y las variables locales
  - Propiedades similares a la memoria global

2.2. Modelo de memoria

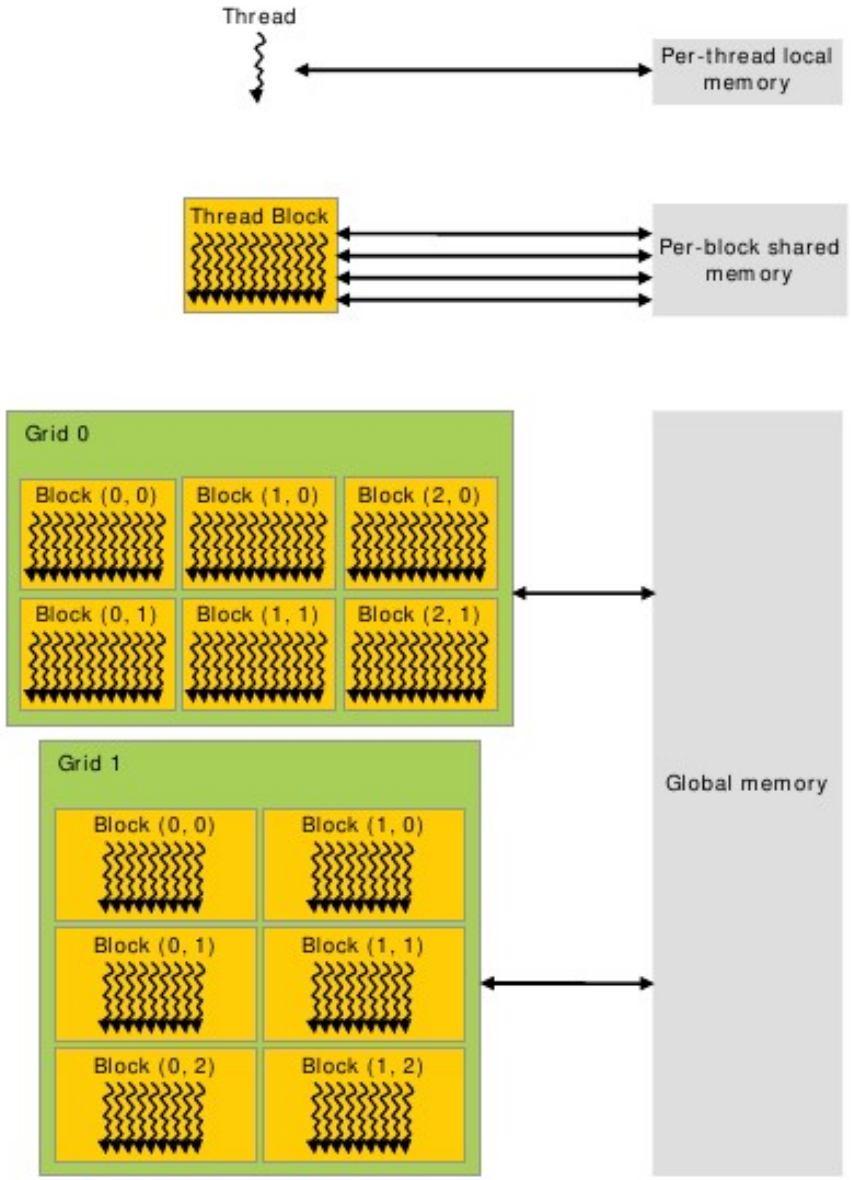
- **Memoria constante (*constant memory*):**
  - Todos los *threads* de un *warp* pueden leer el mismo valor de la memoria constante simultáneamente en un ciclo de reloj
  - Tiempo de acceso similar a los registros
  - Sólo admite operaciones de lectura

- **Memoria de texturas (*texture memory*):**
  - Explota localidad espacial con vectores 1D ó 2D
  - Tiempo de acceso elevado pero menor que memoria global
  - Sólo admite operaciones de lectura

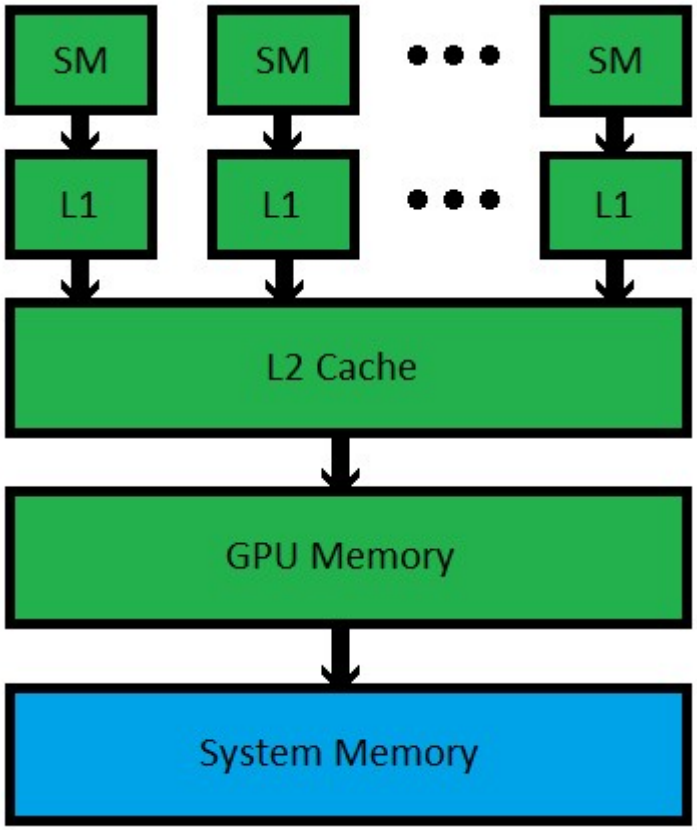
2.2. Modelo de memoria



2.2. Modelo de memoria



2.2. Modelo de memoria



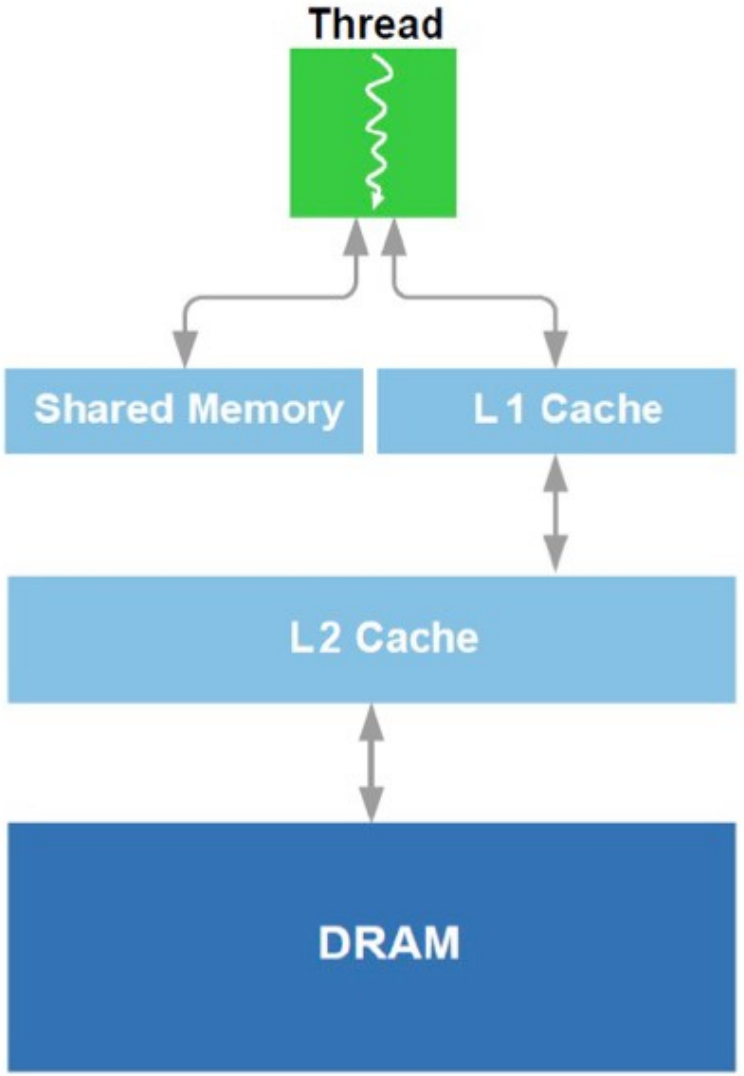
2.2. Modelo de memoria

Configuración SharedMemory vs. L1

- Fermi (GPUcc 2.x): primera GPU que ofrece una caché L1 que combina con la memoria compartida en proporción 3:1 o 1:3

- Total de 64 Kbytes por cada multiprocesador. Por defecto:
  - 48KB de shared memory
  - 16 KB para cache L1

(a partir de la 3.7, hay 128 Kb por Multiprocesador, pero se mantiene el límite de 48 Kb de memoria compartida por bloque)



2.2. Modelo de memoria

Configuración SharedMemory vs. L1

- **`cudaDeviceSetCacheConfig (options):`**
  - establece configuración para todos los kernels

- **`cudaFuncSetCacheConfig (kernel,options):`**
  - establece configuración para un kernel específico

- En ambas funciones, el parámetro **`options`**:
  - `cudaFuncCachePreferShared`: shared memory is 48 KB
  - `cudaFuncCachePreferEqual`: shared memory is 32 KB
  - `cudaFuncCachePreferL1`: shared memory is 16 KB
  - `cudaFuncCachePreferNone`: no preference

Contenidos

2.3. Ejemplo 0: deviceQuery

- /home/usr/local/cuda/extras/demo_suite/deviceQuery
- Compilar ejemplo:

```
make
```

- Ejecutar ejemplo:

```
./deviceQuery
```

- Editar ejemplo:

```
vim|joe deviceQuery.cu
```

  - Llamadas a CUDA *Runtime* API

deviceQuery (marte): Geforce GTX 480 (Fermi (2.x))

```
Device 0: "GeForce GTX 480"

  CUDA Driver Version / Runtime Version          5.0 / 5.0
  CUDA Capability Major/Minor version number:    2.0
  Total amount of global memory:                 1536 MBytes (1610285056 bytes)
  (15) Multiprocessors x ( 32) CUDA Cores/MP:    480 CUDA Cores
  GPU Clock rate:                                1401 MHz (1.40 GHz)
  Memory Clock rate:                             1848 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 786432 bytes
  Max Texture Dimension Size (x,y,z)             1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)
  Max Layered Texture Size (dim) x layers        1D=(16384) x 2048, 2D=(16384,16384) x 2048
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 65535
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
    Support host page-locked memory mapping:      Yes
    Device supports Unified Addressing (UVA):     Yes
  Device PCI Bus ID / PCI location ID:           2 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.0, CUDA Runtime Version = 5.0, NumDevs = 1, Device0
    = GeForce                                                                          GTX 480
```

deviceQuery (saturno): Tesla K20c (Kepler (3.x))

```
Device 0: "Tesla K20c"
  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 4800 MBytes (5032706048 bytes)
  (13) Multiprocessors, (192) CUDA Cores/MP:     2496 CUDA Cores
  GPU Max Clock rate:                            706 MHz (0.71 GHz)
  Memory Clock rate:                             2600 Mhz
  Memory Bus Width:                              320-bit
  L2 Cache Size:                                 1310720 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 2 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.5, CUDA Runtime Version = 7.5, NumDevs = 1,
     Device0 = Tesla K20c
Result = PASS
```

deviceQuery (venus): Geforce GT 640 (Kepler (3.x))

```
Device 1: "GeForce GT 640"
  CUDA Driver Version / Runtime Version          11.2 / 8.0
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 981 MBytes (1028849664 bytes)
  ( 2) Multiprocessors, (192) CUDA Cores/MP:     384 CUDA Cores
  GPU Max Clock rate:                            1046 MHz (1.05 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 524288 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 2 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
> Peer access from Quadro P2200 (GPU0) -> GeForce GT 640 (GPU1) : No
> Peer access from GeForce GT 640 (GPU1) -> Quadro P2200 (GPU0) : No
```

deviceQuery (venus): Quadro P2200 (Pascal (6.x))

```
Device 0: "Quadro P2200"
  CUDA Driver Version / Runtime Version          11.2 / 8.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 5059 MBytes (5304745984 bytes)
  (10) Multiprocessors, (128) CUDA Cores/MP:     1280 CUDA Cores
  GPU Max Clock rate:                            1493 MHz (1.49 GHz)
  Memory Clock rate:                             5005 Mhz
  Memory Bus Width:                              160-bit
  L2 Cache Size:                                 1310720 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 3 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

deviceQuery (venus): Quadro P2200 (Pascal (6.x))

```
Device 0: "Quadro P2200"
  CUDA Driver Version / Runtime Version          11.2 / 8.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 5059 MBytes (5304745984 bytes)
  (10) Multiprocessors, (128) CUDA Cores/MP:     1280 CUDA Cores
  GPU Max Clock rate:                            1493 MHz (1.49 GHz)
  Memory Clock rate:                             5005 Mhz
  Memory Bus Width:                              160-bit
  L2 Cache Size:                                 1310720 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 3 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```