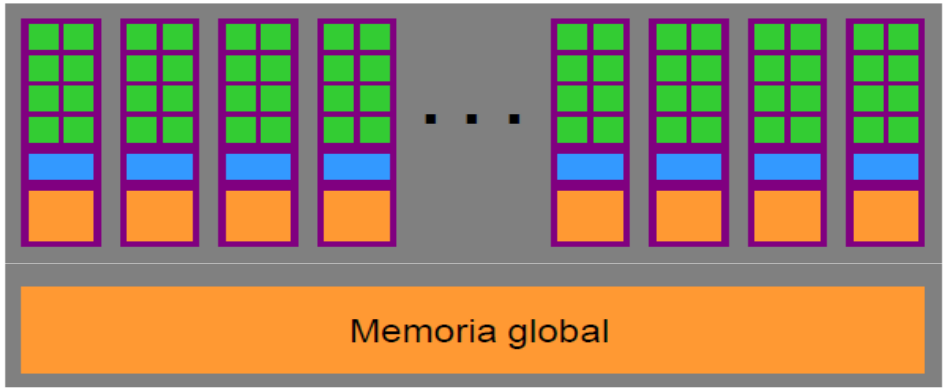
GPU-CUDA

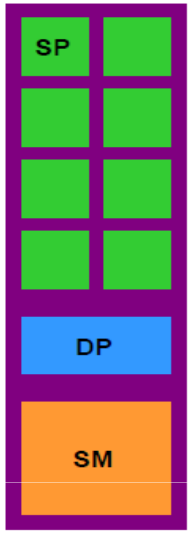
GPGPU
(GENERAL PURPOSE COMPUTING ON GRAPHICS PROCESSING UNITS)

- Técnica GPGPU consiste en el uso de las GPU para resolver problemas computacionales de todo tipo aparte de los relacionados con el procesamiento de gráficos que son los que una GPU normalmente resuelve.

ARQUITECTURA GPU

Conjunto de Streaming Multiprocessors (MP)



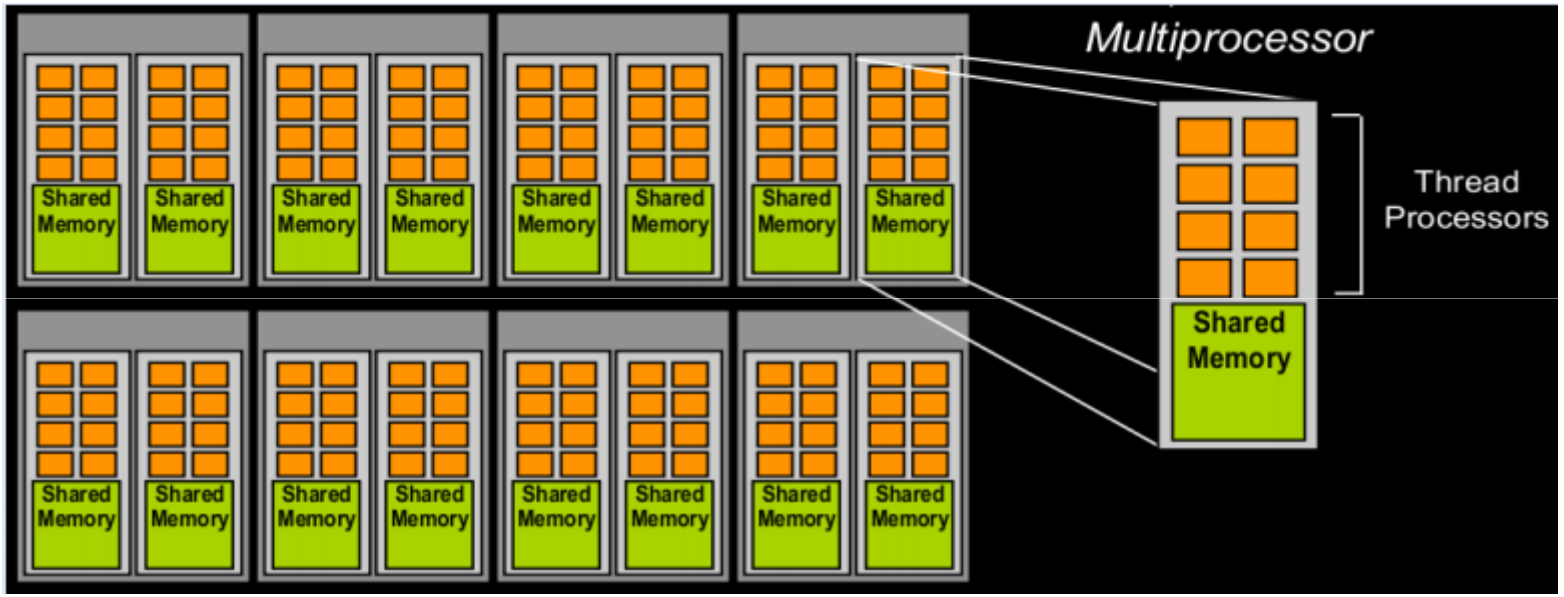


8 Scalar Processors (SP)
1 Unidad de Doble Precisión (DP)
16 KB de Memoria Compartida (SM)
8-16 K Registros

SERIES 8 (G80)

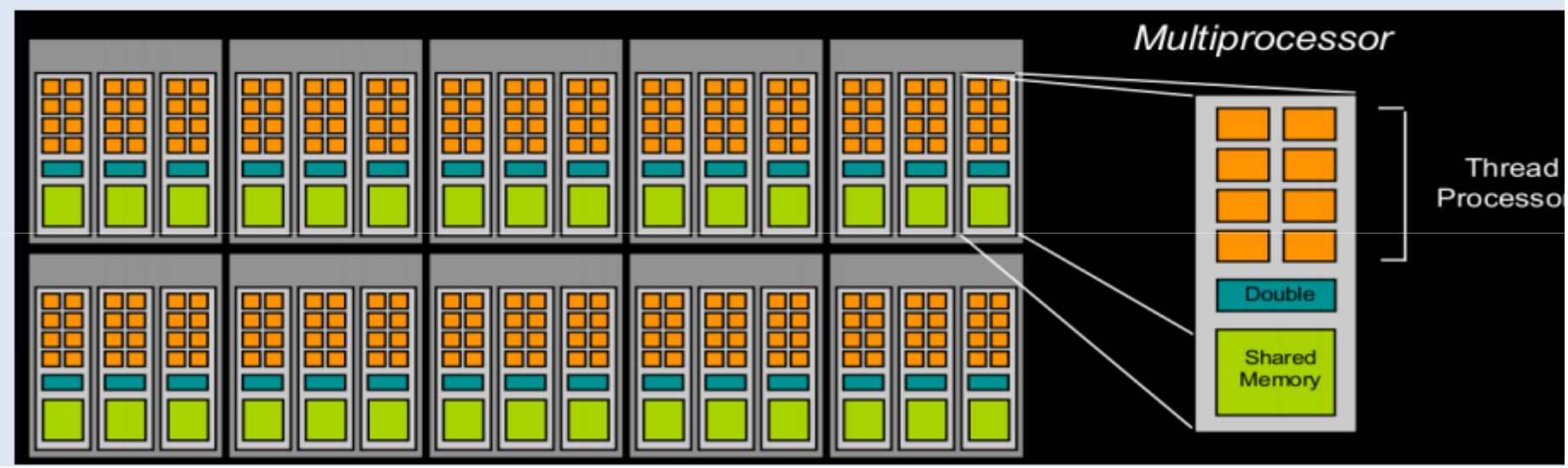
16 multiprocesadores SM cada uno con 8 núcleos

Memoria compartida de 16 K



SERIE 200 (GT200)

30 SMs cada uno con 8 núcleos, una unidad de doble precisión y memoria compartida de 16K



FERMI (GT400)

512 cores

16 multiprocs.

Cada multiprocs. con 32 nuúleos y 16 Unidades de doble precisión.

64 KB. de SRAM a repartir entre memoria compartida y Cache L1.

COMPUTE CAPABILITY

Para hacer uso óptimo de las GPUs, es necesario conocer diferentes características de la tarjeta.

Nvidia utiliza un formato estandarizado para especificar estas características denominado **compute capabilities** (CC, capacidades de computación).

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 son de compute capability 1.0.
- Posteriores GPUs pertenecen a las compute capabilities 2.0 2.1 (pre-Kepler) ,3.0, 3.5, 3.7,5.0, 5.2

¿QUÉ ES CUDA?

Compute Unified Device Architecture (CUDA)

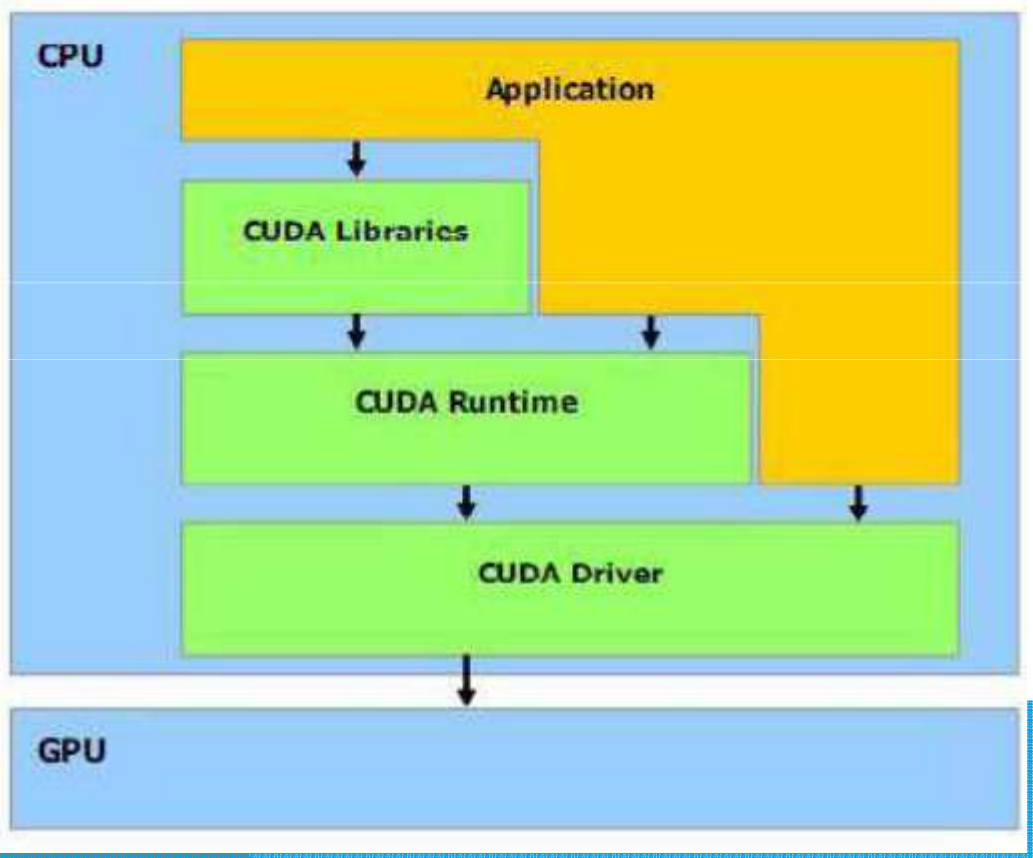
- Es una arquitectura de computación paralela para el cómputo de problemas de propósito general GPGPU) diseñada por Nvidia.
- Está enfocada al **cálculo masivamente paralelo** y las capacidades de procesamiento que brinda las tarjetas gráficas de Nvidia.
- Permite programar el dispositivo a través de extensiones de lenguajes de programación estándar (C , C++y Fortran).
- Está disponible para las tarjetas gráficas GeForce de la serie 8 en adelante.
- Es compatible con Linux de 32/64 bits y Windows XP (y sucesores) de 32/64 bits.

ARQUITECTURA CUDA

- La arquitectura CUDA se puede definir como una arquitectura SIMT (Single Instruction, Multiple Thread; instrucción única, múltiples hilos), análoga a la arquitectura SIMD.
- La idea fundamental es que cada hilo ejecute la misma instrucción o función sobre un dato diferente en paralelo.

ARQUITECTURA CUDA

Tiene una pila de capas de software que incluye bibliotecas, el CUDA Runtime y el CUDA Driver.



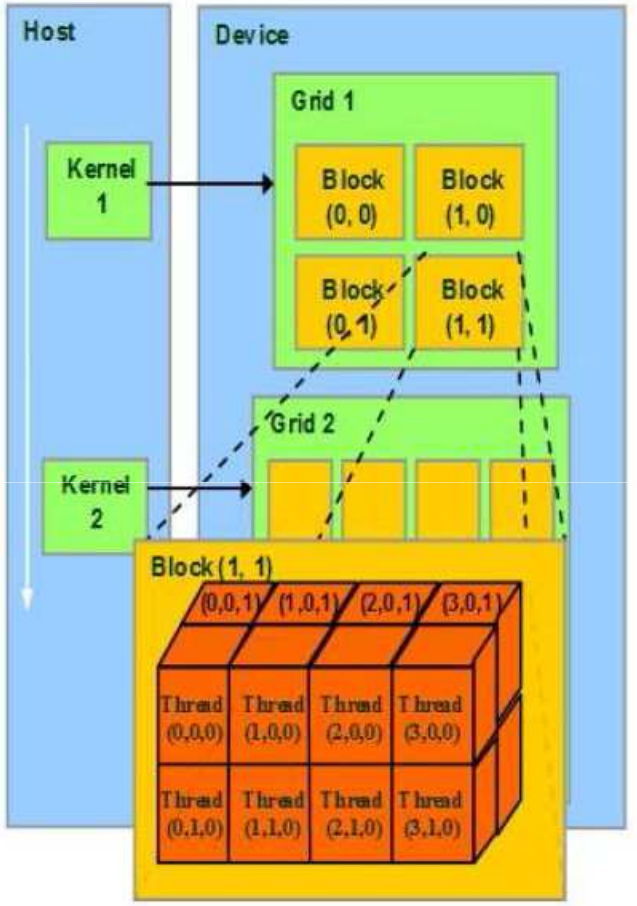
MODELO DE EJECUCIÓN

MODELO DE EJECUCIÓN

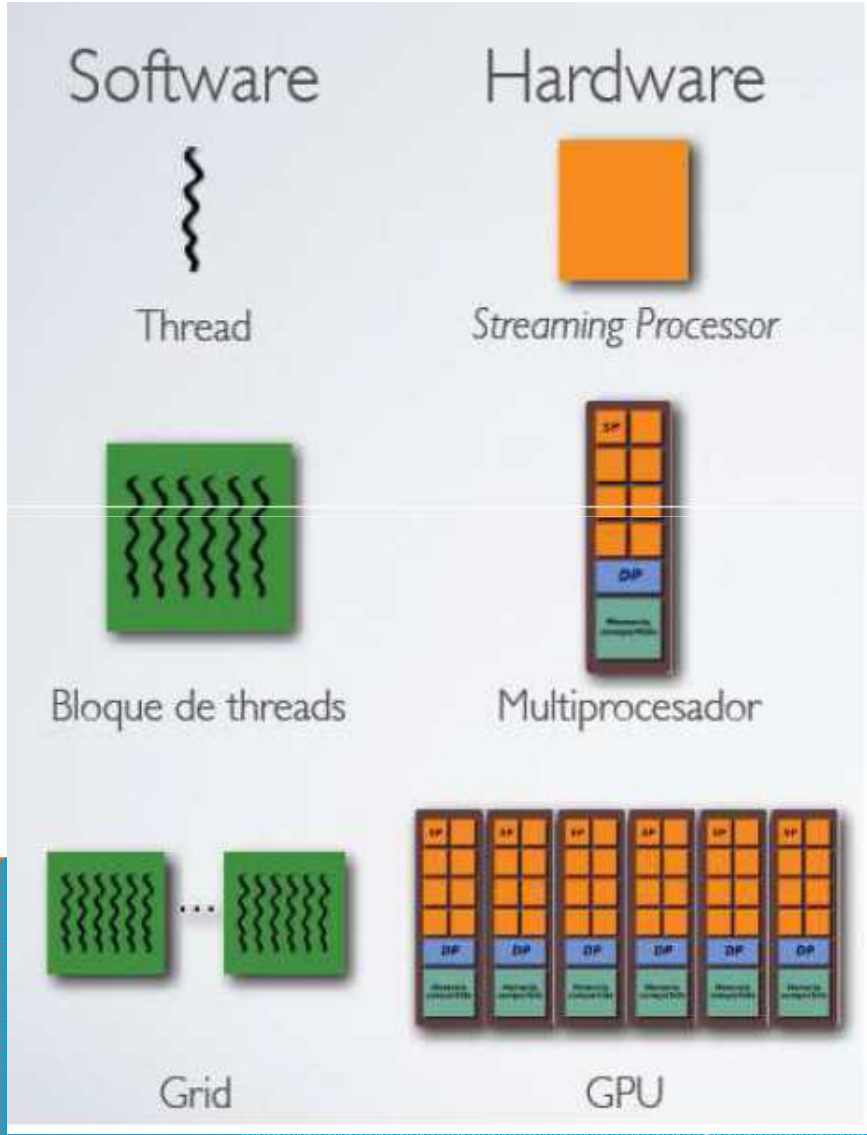
- En el modelo de ejecución de CUDA cada multiprocesador ejecuta el mismo programa o función pero sobre distintos datos (arquitectura SIMT).
- Este paradigma de programación se conoce como SPMD (Single Program Multiple data).

MODELO DE EJECUCIÓN

- Los programas que se 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 se organizan en un grid.
  - Los bloques pueden ser: 1D, 2D o 3D (3D desde Fermi)
  - La Grid puede ser 1D o 2D



MODELO DE EJECUCIÓN

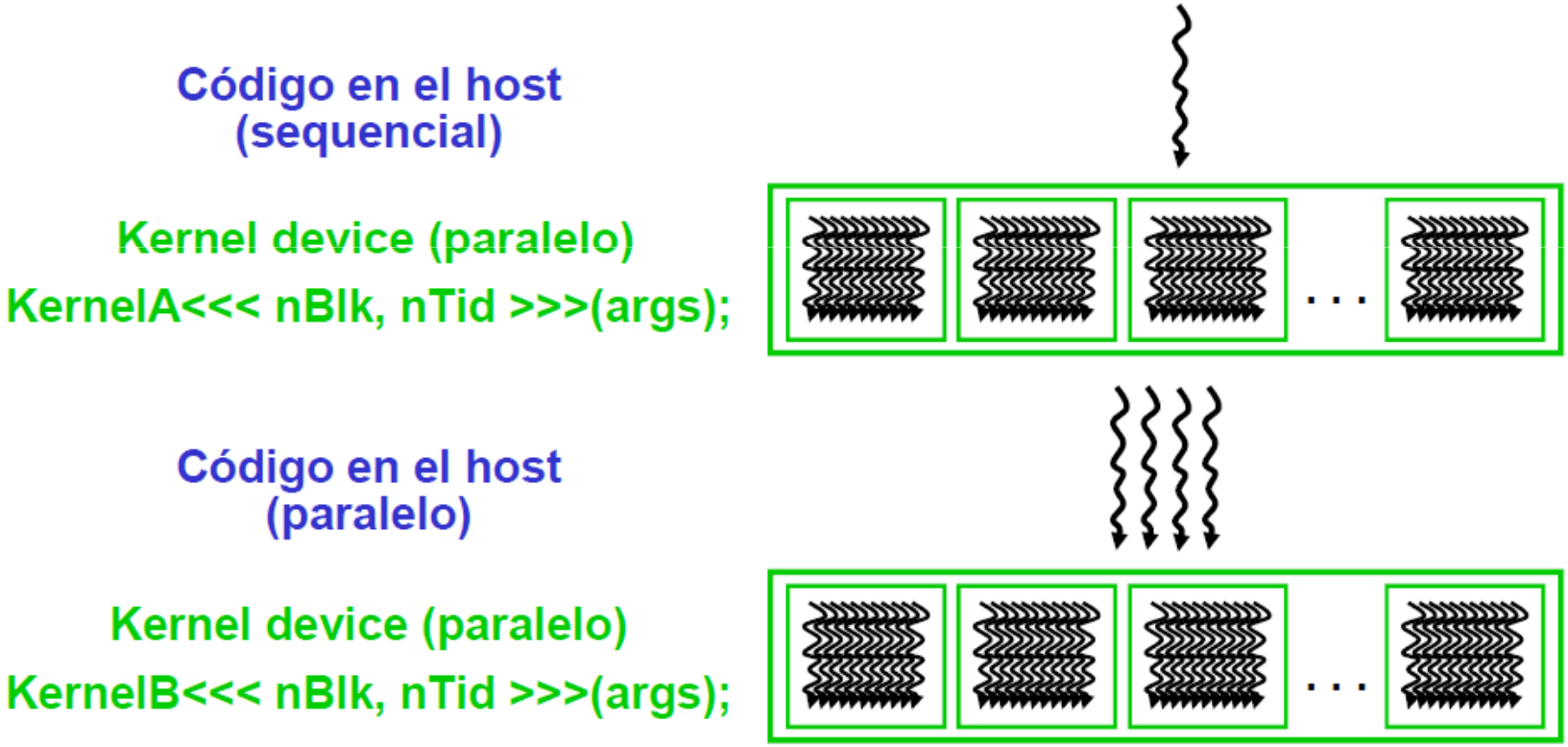


MODELO DE PROGRAMACIÓN

MODELO DE PROGRAMACIÓN

Se integra código del *host* y del *device*

- El *device* es un GPU, opera como un coprocesador .



COMPILACIÓN

nvcc prog.cu –o prog

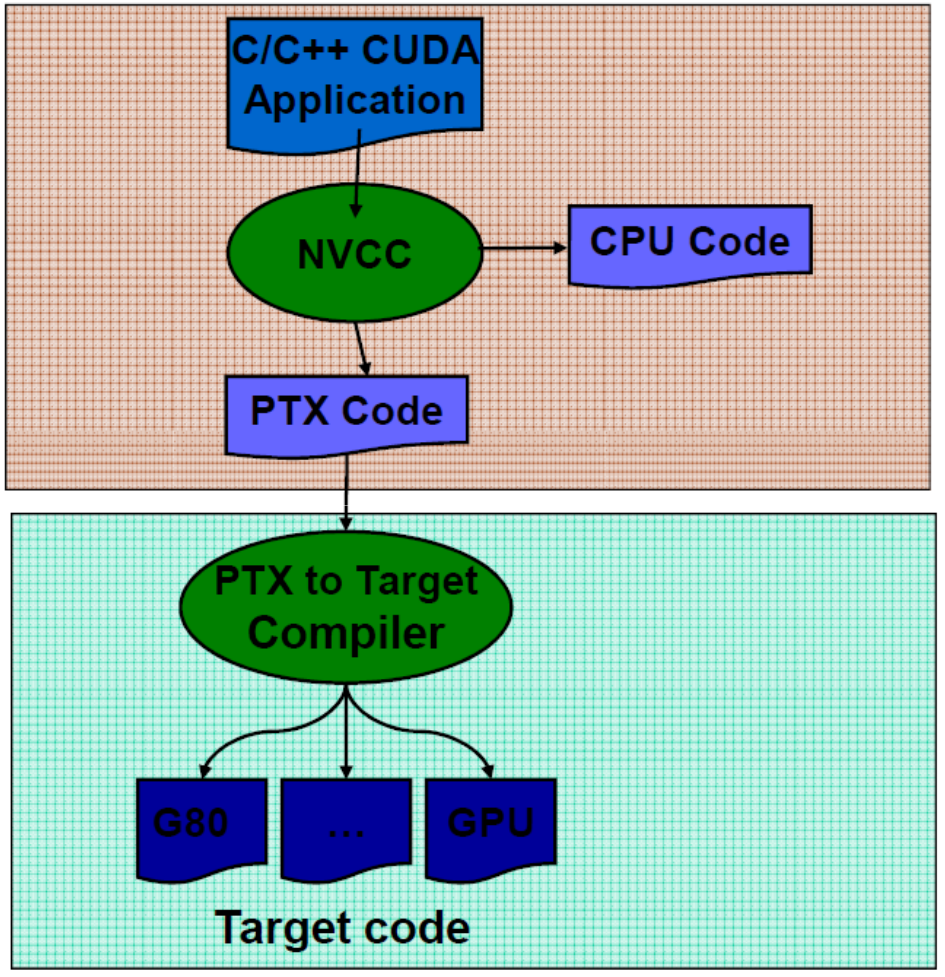
Salidas:

Código C (código del host)

- Se compila con una herramienta tradicional (automáticamente).

Código PTX (Parallel Thread eXecution)

- Directamente código objeto.



EJEMPLO 1
PRIMER PROGRAMA

Hola Mundo

MODIFICADORES DE FUNCIÓN

¿Dónde se ejecuta la función?

__device__ La función debe ejecutarse en el dispositivo

- Sólo puede ser llamada por el propio dispositivo
- Recursividad no soportada
- No pueden declararse variables estáticas dentro de la función

__global__La función es un kernel que debe ejecutarse en el dispositivo

- Sólo puede ser llamada por el host
- Recursividad no soportada
- No pueden declararse variables estáticas dentro de la función
- La función debe devolver siempre void

__host__La función debe ejecutarse en el host

- Sólo puede ser llamada por el host
- No puede utilizarse junto con __global__

FUNCIONES

- cudaMalloc(void ** *devPtr*,size_t *size*)
- cudaFree(void * *devPtr*)
- cudaMemcpy(void * *dst*,const void * *src*,size_t *count*,enum cudaMemcpyKind *kind*)
- cudaMemcpyKind
  - cudaMemcpyHostToDevice
  - cudaMemcpyDeviceToHost
  - cudaMemcpyDeviceToDevice

EJEMPLO 2

Realizar un programa donde el dispositivo sume dos números enteros

NOTA: Se utilizará un bloque con un hilo