Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

CC3501:
Computación Gráfica, Visualización y Modelación para Ingenieros

Otoño 2012

GPU y CUDA

Daniel Calderon S.

Contenido

- Introducción a las GPU
- Concepto GPGPU
- Arquitectura CUDA
- Aplicaciones
- ¿Donde y como aprender?
- Conclusiones

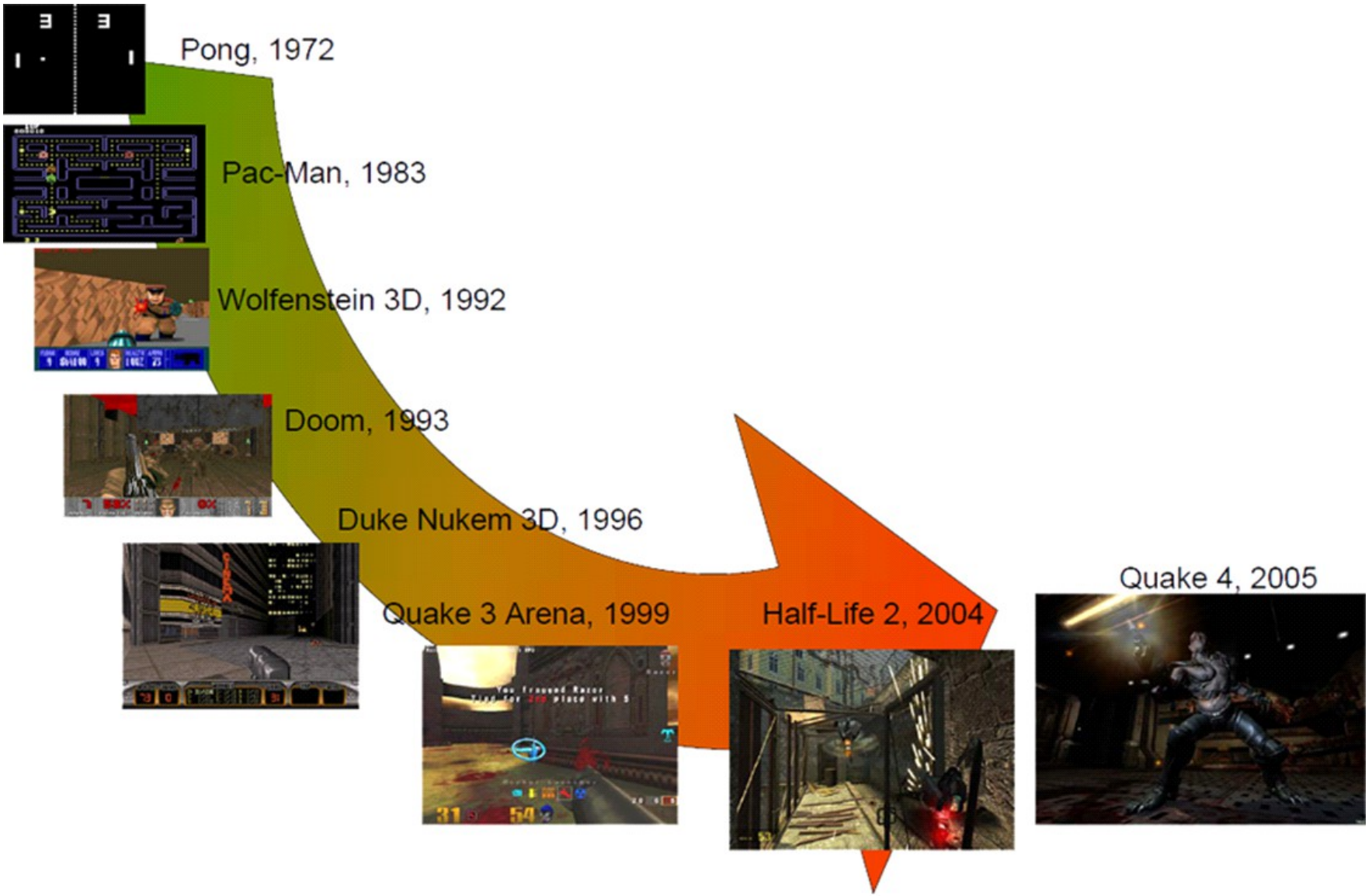
GPU:
Graphics Processing Unit

Origen de las GPU

- Complejidad del procesamiento de los graficos a mostrar en pantalla.
- Procesamiento de cada uno de los vértices y pixeles.
- Demasiado trabajo para la CPU, ralentizando las “verdaderas” aplicaciones.

- Solución:
  Desarrollo de hardware específico, la GPU.

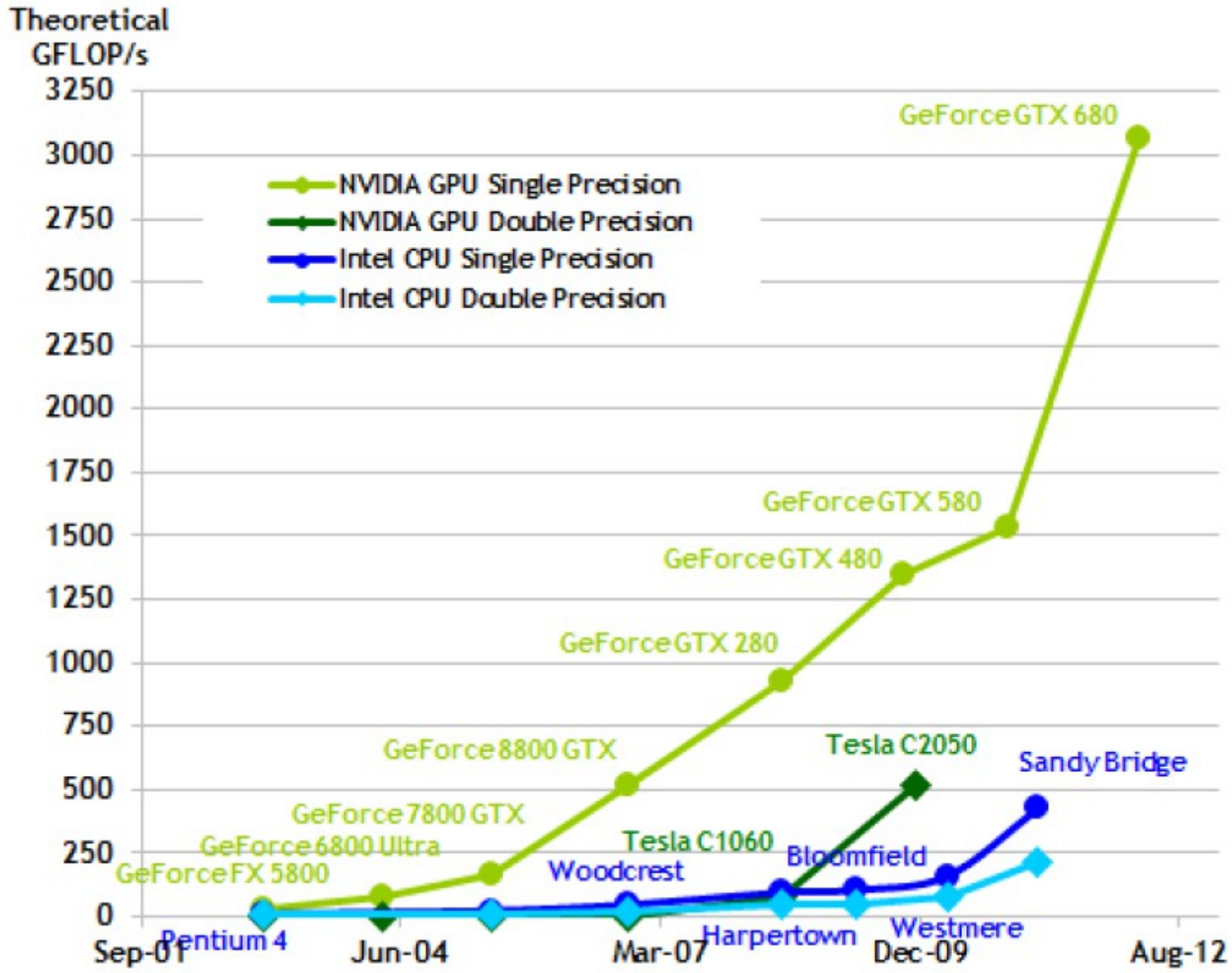
Evolución de las GPU



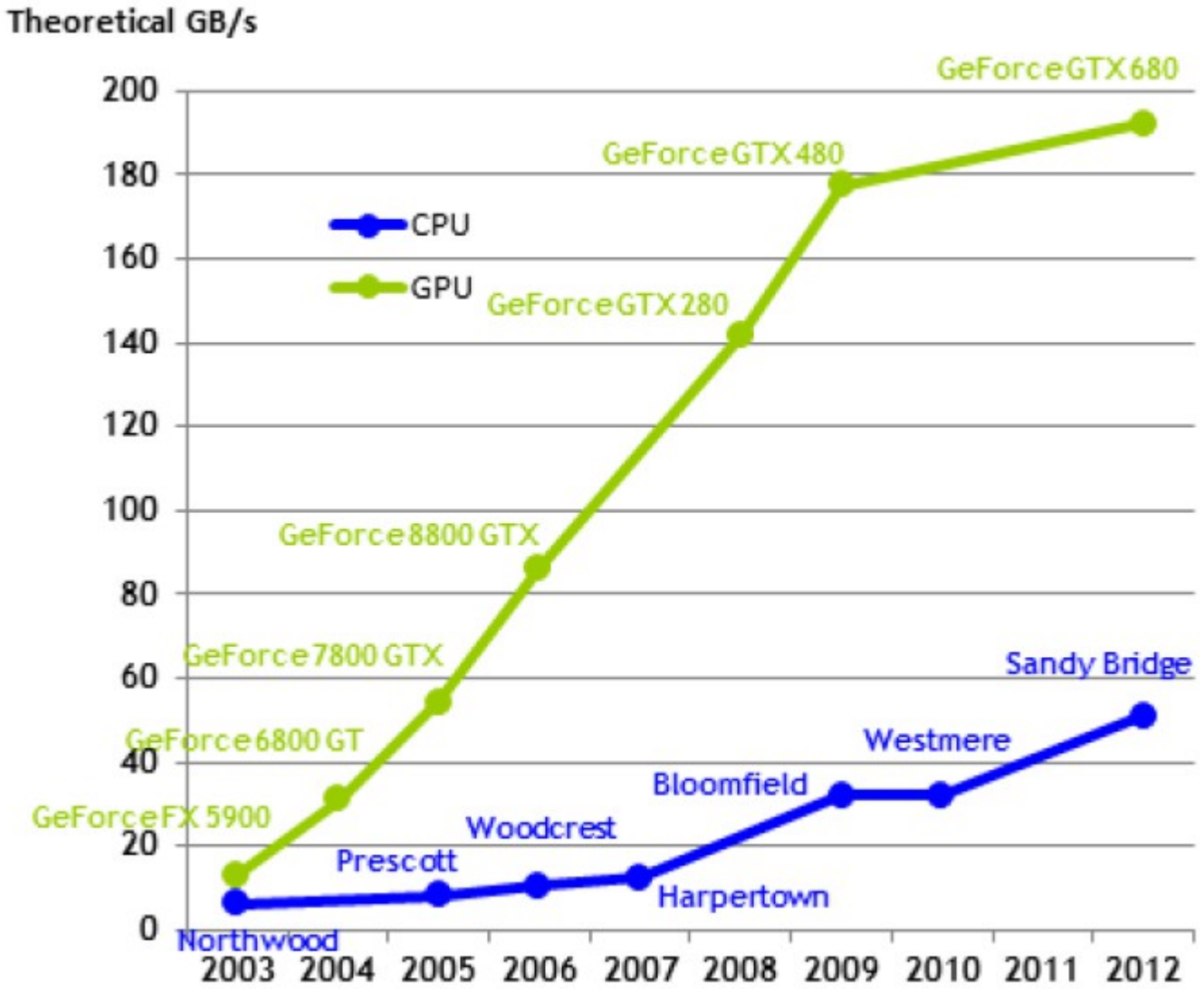
Evolución de las GPU

- Debido a la alta demanda de gráficos 3D de alta definición y en tiempo real, las GPU han evolucionado en una unidad altamente paralela, con soporte de multithread y con procesadores de varios núcleos.
- Se tiene entonces una enorme potencia de cálculo y un elevadísimo ancho de banda.

Evolución de las GPU

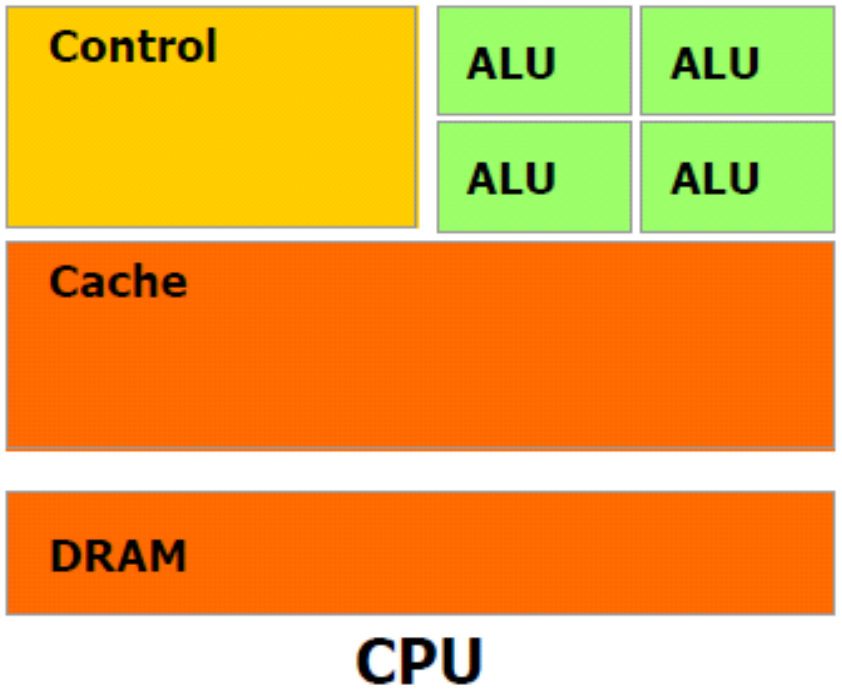


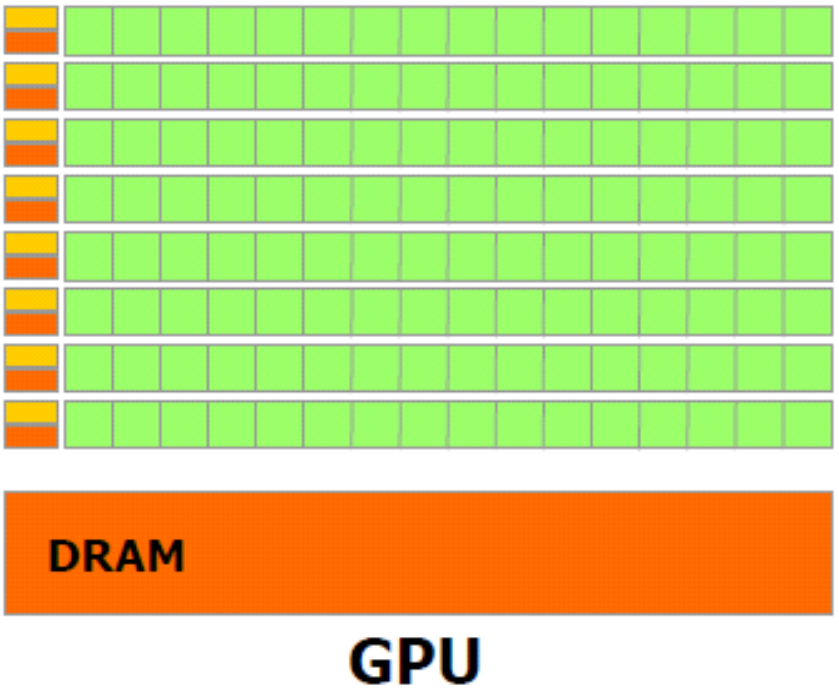
Evolución de las GPU



CPU vs GPU

- Arquitecturas diferentes





CPU vs GPU

- CPU
  - Acceso aleatorio a memoria
    - Creación de estructuras de datos complejas, con punteros a posiciones arbitrarias en memoria.
- GPU
  - Acceso a memoria mucho más restringido
    - Procesador de vértices
    - Procesador de pixeles

CPU vs GPU

- Procesador de vértices
  - Modelo Scatter
    - Leer posición predeterminada, escribir una o varias posiciones aleatorias.
- Procesador de pixeles
  - Modelo Gather
    - Leer posiciones aleatorias, escribir en posición determinada.

GPGPU:
General-Purpose Computing on Graphics Processing Units

GPGPU

- Concepto reciente que intenta estudiar y analizar las capacidades de cómputo de una GPU fuera del ámbito gráfico.
- La idea surge de:
  - Bajo costo en relación a su potencia de cálculo
  - Gran nivel de paralelismo
  - Optimización para cálculos en punto flotante

Contraparte de GPGPU

- Falta de precisión
  - Para procesar un número de punto flotante, se utilizan:
    - 2 a 4 bytes en GPU.
    - 4 a 8 bytes en CPU.
- Altamente específicos para funciones del ámbito gráfico.
- Falta de continuidad en las arquitecturas. Algoritmos dejan de funcionar de manera óptima.

Programando una GPU

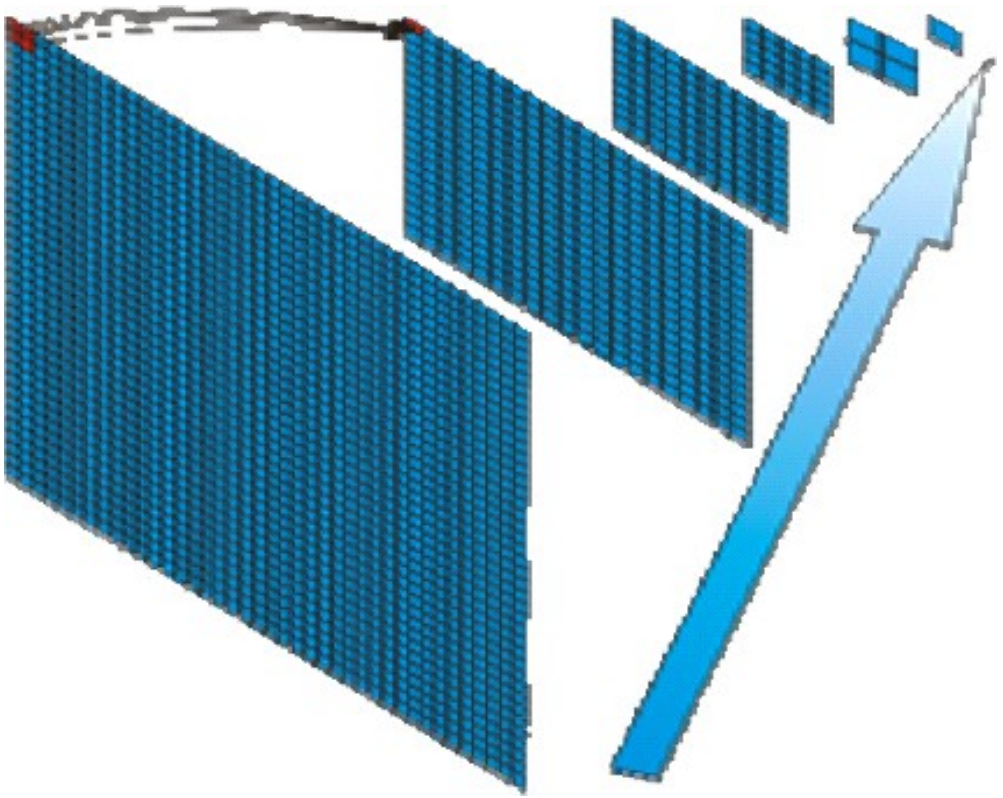
- Ensamblador (lenguaje propio de la arquitectura)
- APIs para fines gráficos
  - GLSL, OpenGL
  - HLSL, Microsoft
  - CG, nVidia
- APIs para propósito general
  - C o FORTRAN con extensiones CUDA (en nVidia)
  - OpenCL, grupo Khronos
  - Direct Compute, Microsoft

Diseñar un algoritmo GPGPU

- Se deben adaptar los accesos a memoria y las estructuras de datos a las caracteríticas de la GPU.
- Sólo es útil para algoritmos altamente paralelizables y que no requieran estructuras de gran complejidad.
- Utilizar lenguajes para fines gráficos es altamente complejo.

Reducción:
Una estrategia de paralelización

- Sumas, multiplicaciones, primedio, máximo, mínimo, etc...
- Disminuye el orden del algoritmo de lineal a logarítmico.



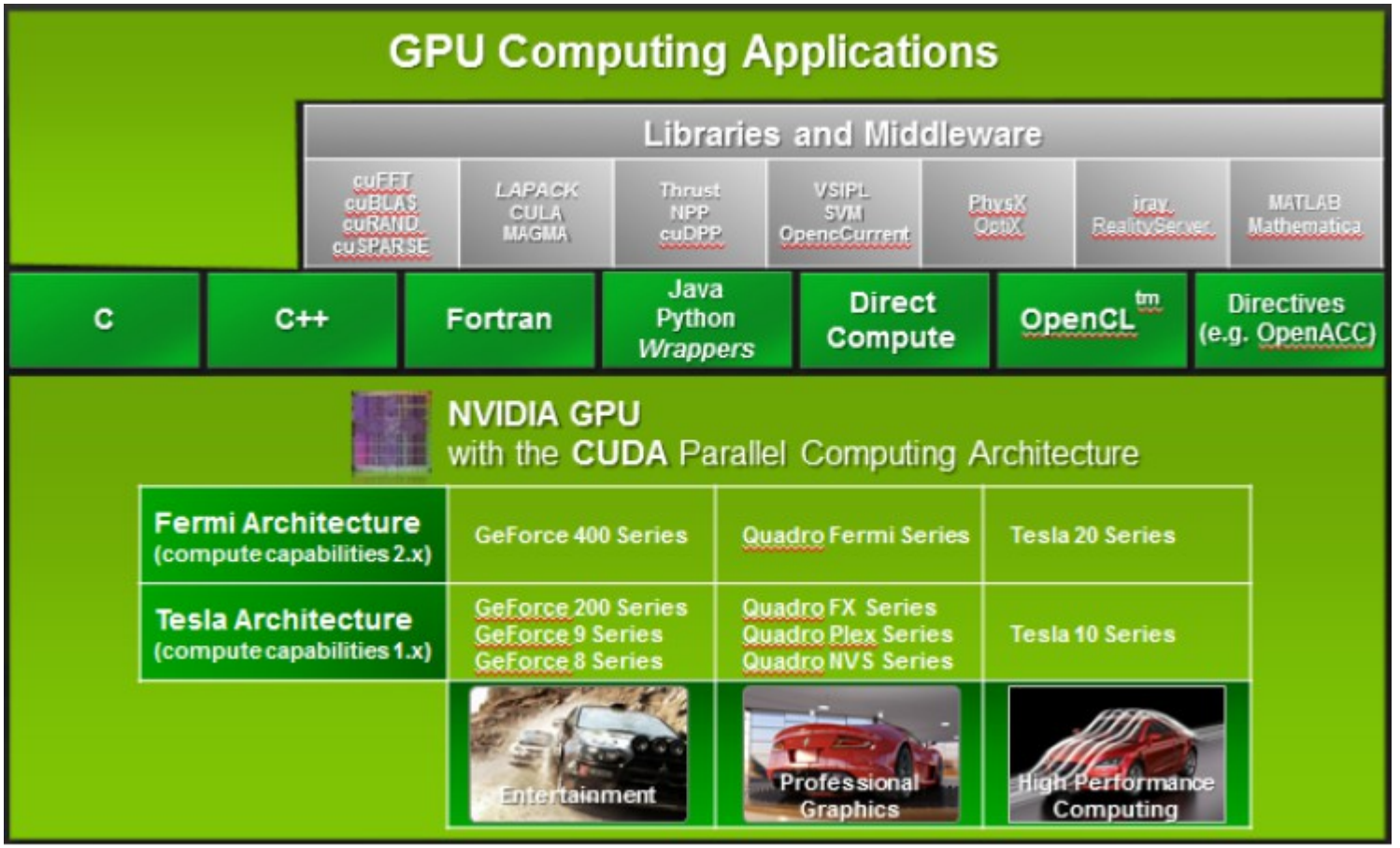
CUDA:
Compute Unified Device Architecture

CUDA

- Presentado en Noviembre de 2006 por nVidia.
- Arquitectura de cómputo paralelo para propósito general.
- Funciona con todas las GPU nVidia desde la serie G8X.
- Se puede programar directamente en C/FORTRAN con algunas características adicionales.

CUDA

- Permite trabajar con GPGPU con distintas APIs:



CUDA

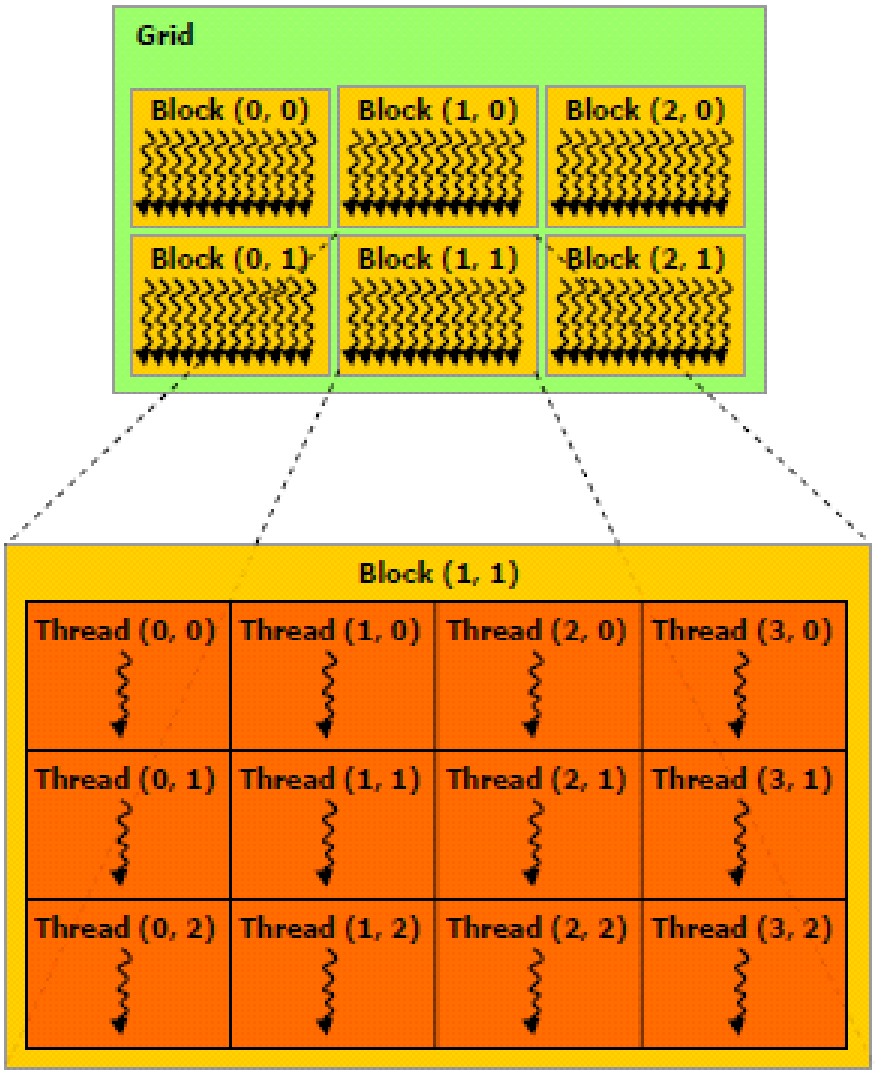
- CUDA se basa en un arreglo de multiprocesadores (SMs, Streaming Multiprocessors).
- Multiprocesador:
  - 8 pocesadores escalares.
  - 2 funciones especiales para trascendentales.
  - Una unidad de instrucciones multithreading.
  - Memoria compartida.

SIMT

- Para manejar cientos de threads, el multiprocesador utiliza una nueva estructura llamada SIMT.
- SIMT, Single-Instruction, Multiple Thread.
- La unidad SIMT del multiprocesador crea, maneja, organiza y ejecuta los theads en grupos de 32 threads paralelos llamados warps.

Organización de Threads

- Los threads se organizan en bloques, los que se distribujen sobre una grilla.
- Cada thread de cada bloque tiene un identificador propio, es decir, un número de thread y de bloque.

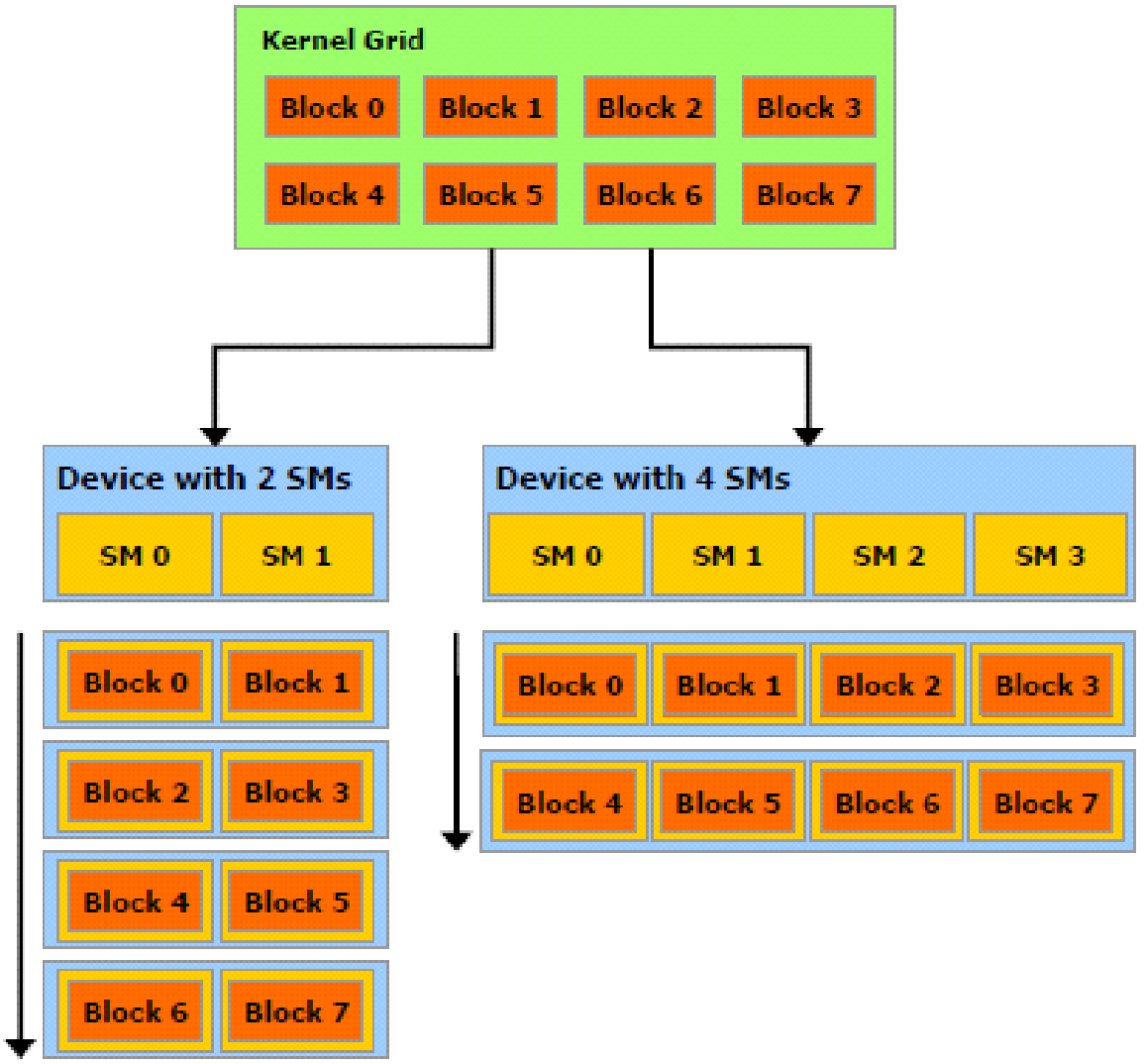


Organización de threads

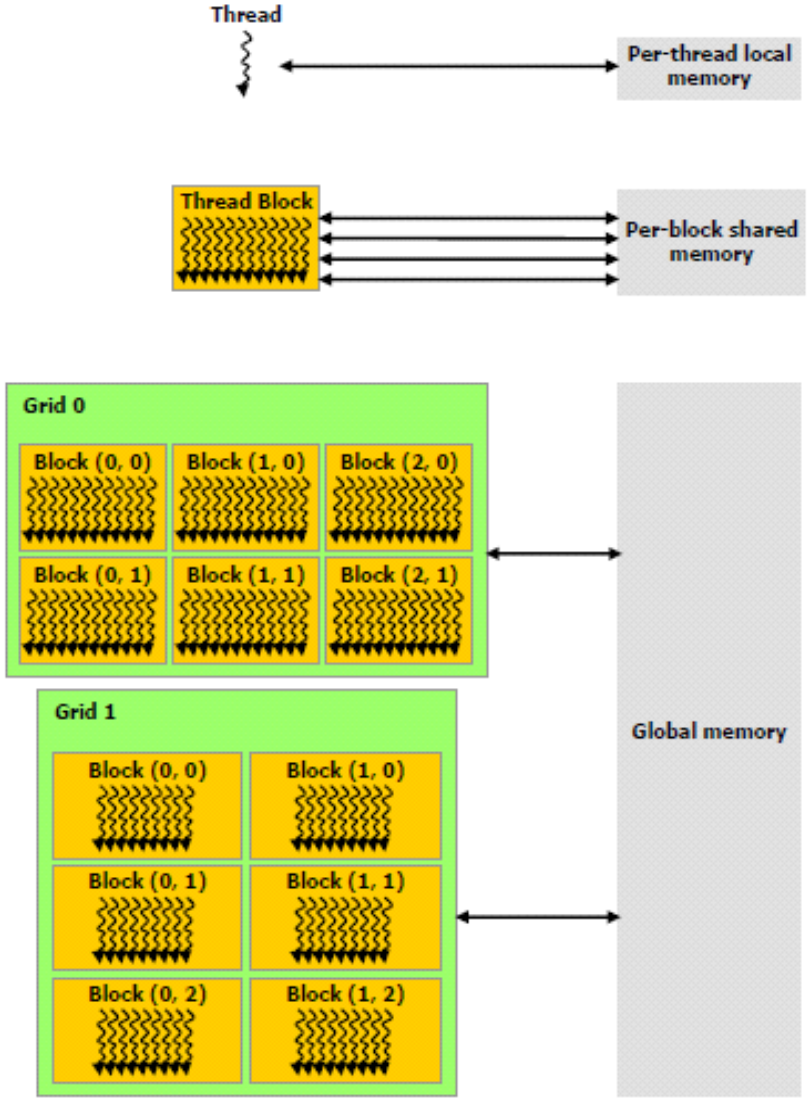
- Los threads de un bloque se ejecutan concurrentemente.
- La ejecución de bloques se puede alternar dependiendo del uso de la memoria compartida.
- Se puede sincronizar la ejecución de los threads estableciendo barreras para compartir información de la memoria compartida.

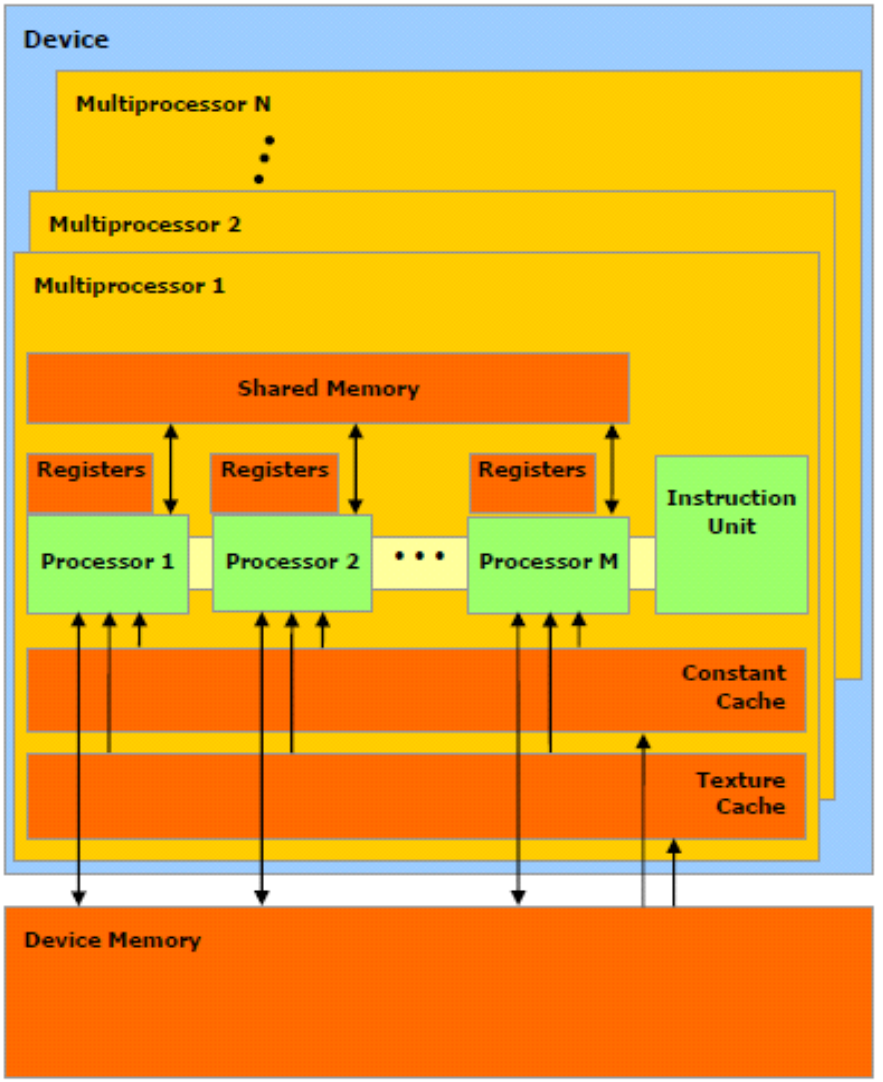
Escalabilidad automática

- Los bloques se organizan distribuyéndo se según las capacidades del dispositivo.



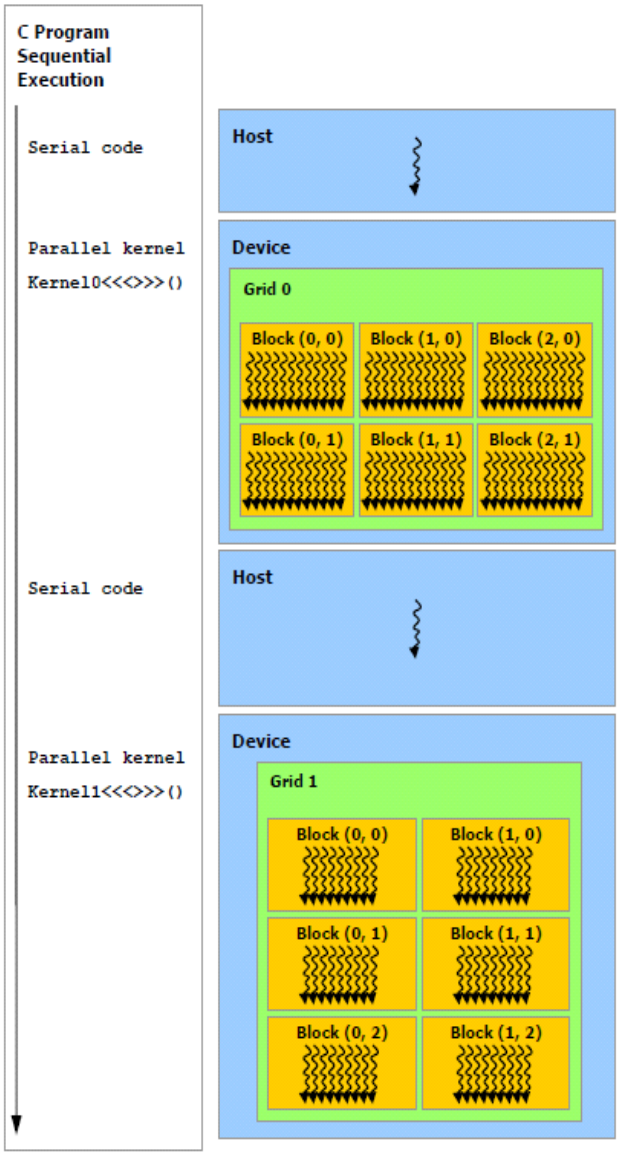
Jerarquía de memoria



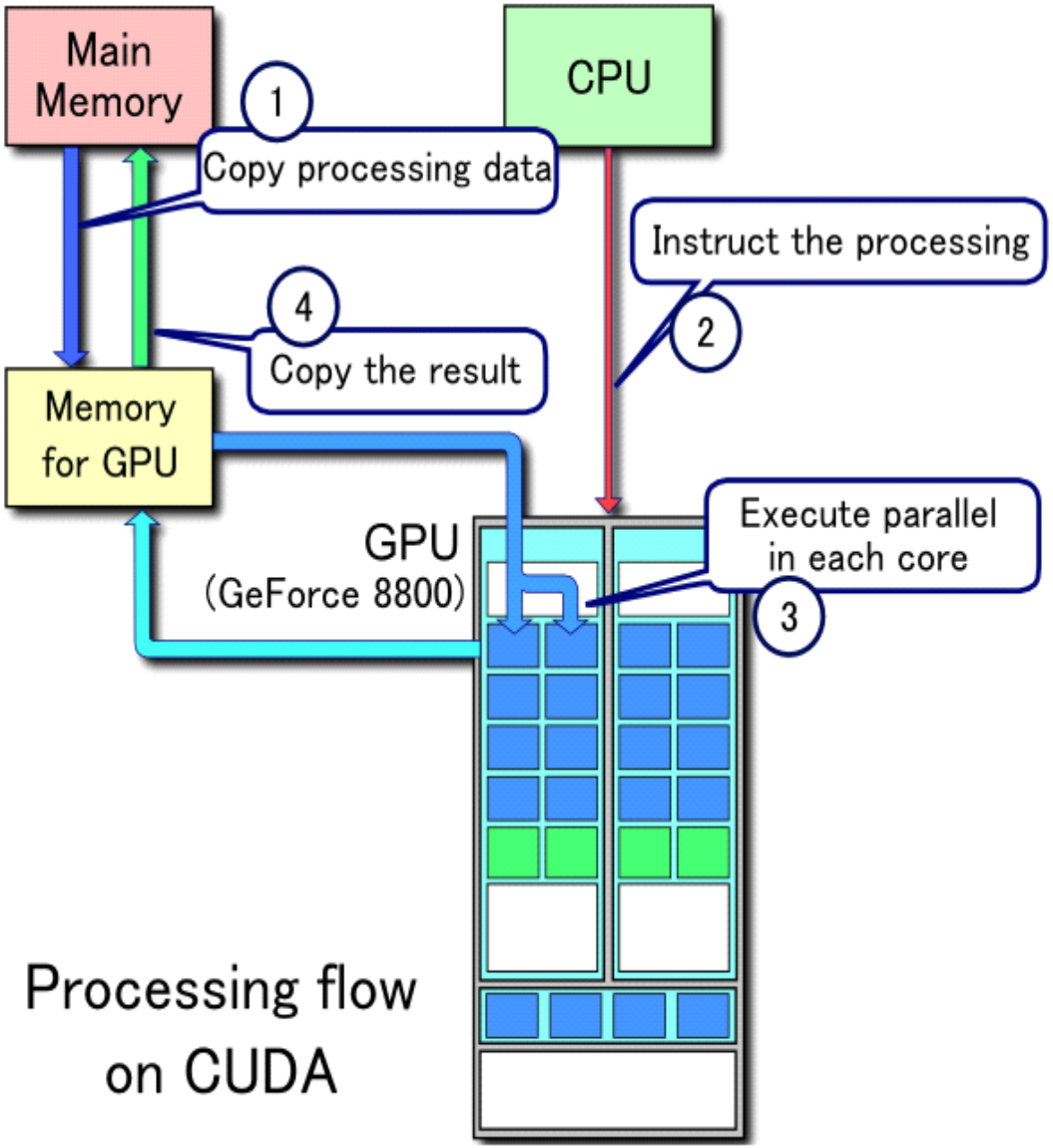


Ejecución de un programa

- Alternanciasimple entre trabajo en CPU y GPU.
- Se puede trabajar paralelamente en CPU y GPU.



Flujo de procesamiento



Especificaciones de una GPU con CUDA

```
There is 1 device supporting CUDA

Device 0: "GeForce 9800 GTX/9800 GTX+"
  Major revision number:                         1
  Minor revision number:                         1
  Total amount of global memory:                 536543232 bytes
  Number of multiprocessors:                     16
  Number of cores:                               128
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:                   16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.67 GHz
  Concurrent copy and execution:                 Yes
```

Programando en CUDA

- Funciones de transferencia de memoria

```
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

- Reserva y liberación de memoria de la GPU

```
float* d_A;
cudaMalloc(&d_A, size);
cudaFree(d_A);
```

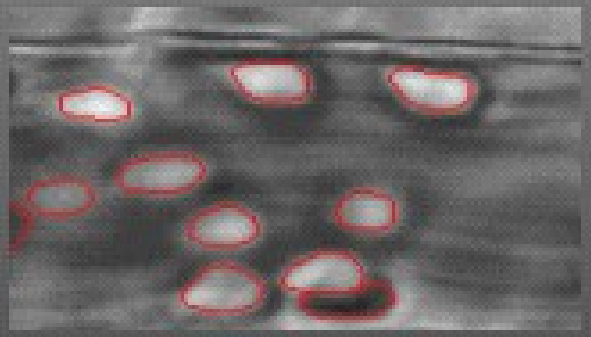
- Llamada a función Kernel (se ejecuta en GPU)

```
int threadsPerBlock = 256;
int blocksPerGrid = (N + threadsPerBlock – 1) / threadsPerBlock;
VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
```

Aplicaciones de GPGPU

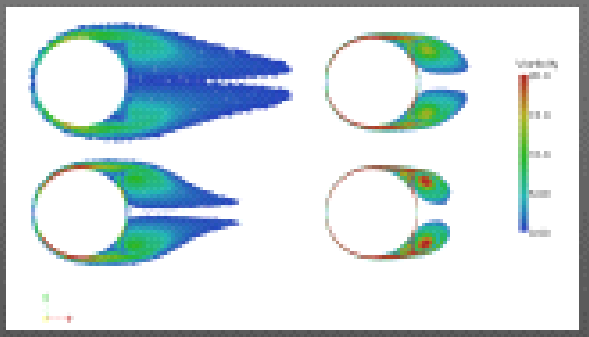
Potenciales aplicaciones

- Procesamiento de imágenes.
- Técnicas de inteligencia
  - Redes Neuronales
  - Teoría de la Información
  - Algorítmos Genéticos
- Medicina
  - Simulación de proteínas
  - Acción de virus
- Matemáticas
  - Resolución de EDPs.



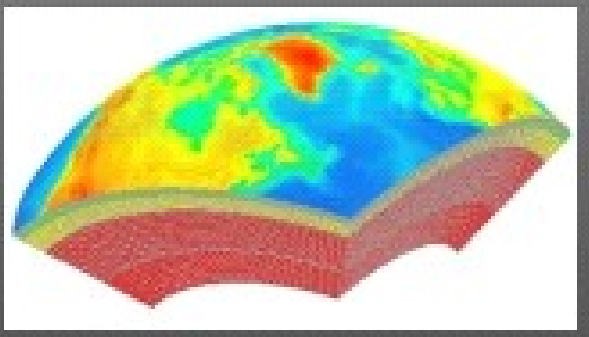
Leukocyte Tracking: ImageJ Plugin

26 x



A GPU-accelerated Boundary Element Method and Vort...

43 x



High-order finite-element seismic wave propagation...

20 x

NeuroSolutions CUDA Add-on

50 x

Incompressible Flow Computations on the NCSA Linco...

Accelerating SQL Database Operations on a GPU with...

70 x

GPU y CUDA

¿Donde y como aprender?

¿Donde y como aprender?

- Descargar e instalar: NVIDIA GPU Computing SDK.

  http://developer.nvidia.com/cuda-downloads
- El SDK incluye la documentación, guías explicativas y ejemplos.
- También se documentan las APIs: OpenCL y DirectCompute

Conclusiones

Conclusiones

- GPGPU es el futuro para el cómputo intensivo.
- Existen muchas aplicaciones que pueden mejorar considerablemente su rendimiento utilizando una GPU.
- CUDA da el paso necesario para poder entenderse de manera simple con la GPU, esto es, sin pasar por los lenguajes gráficos.