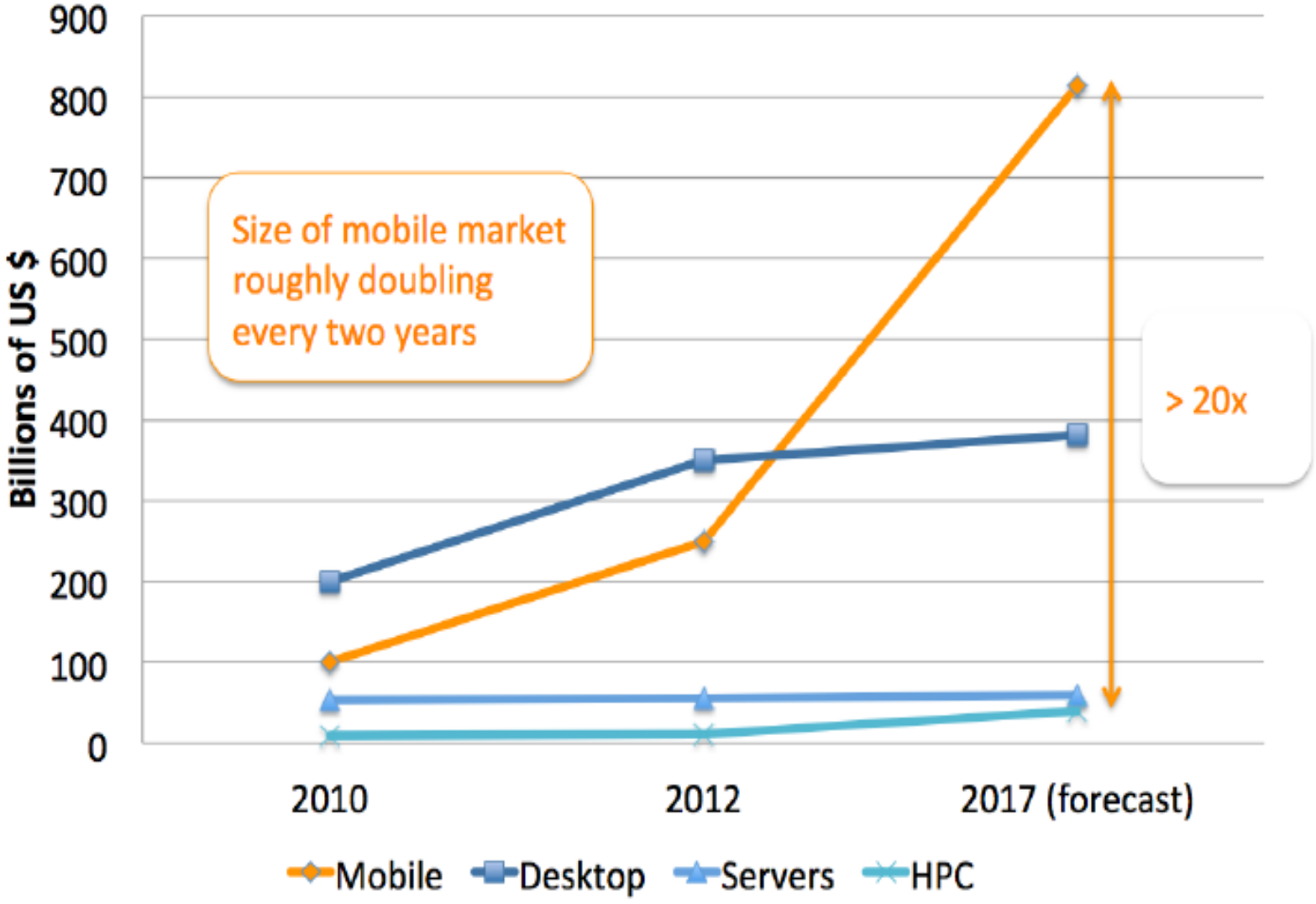
Introducción a la Programación Multicore

Mario Rossainz López
OTOÑO 2022
NRC: 60898

- A new mantra: Power and Energy saving
- In all domains

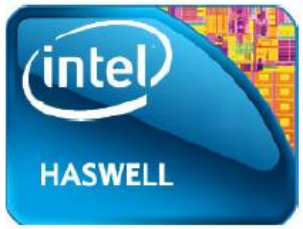
- There is (parallel) live beyond supercomputers:



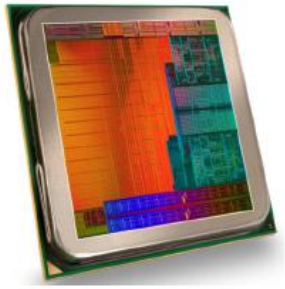
- A partir del año 2005 varios fabricantes como:

- Intel
- IBM
- SUN
- AMD
- Etc…

Intel Haswell



AMD Kaveri

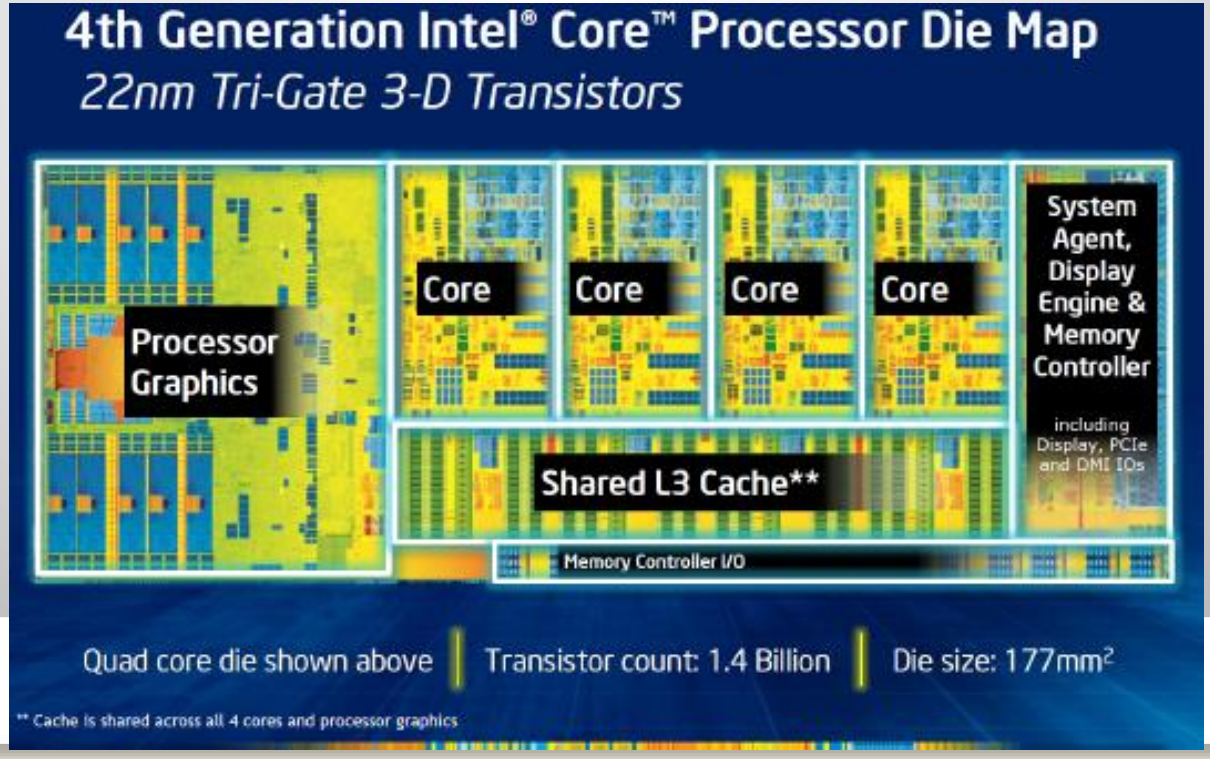


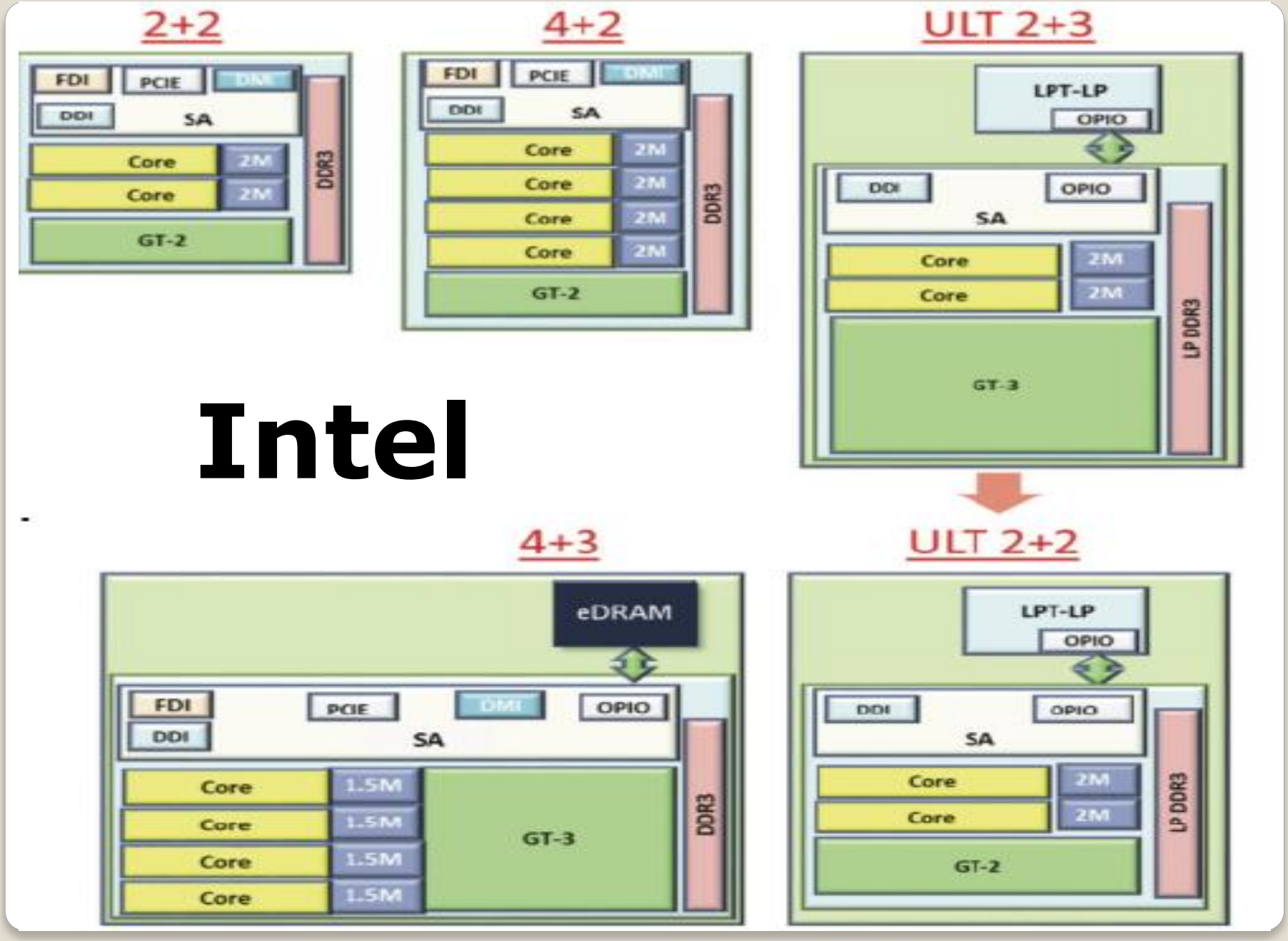
Samsung Exynos 5 Octa

Qualcomm Snapdragon 800

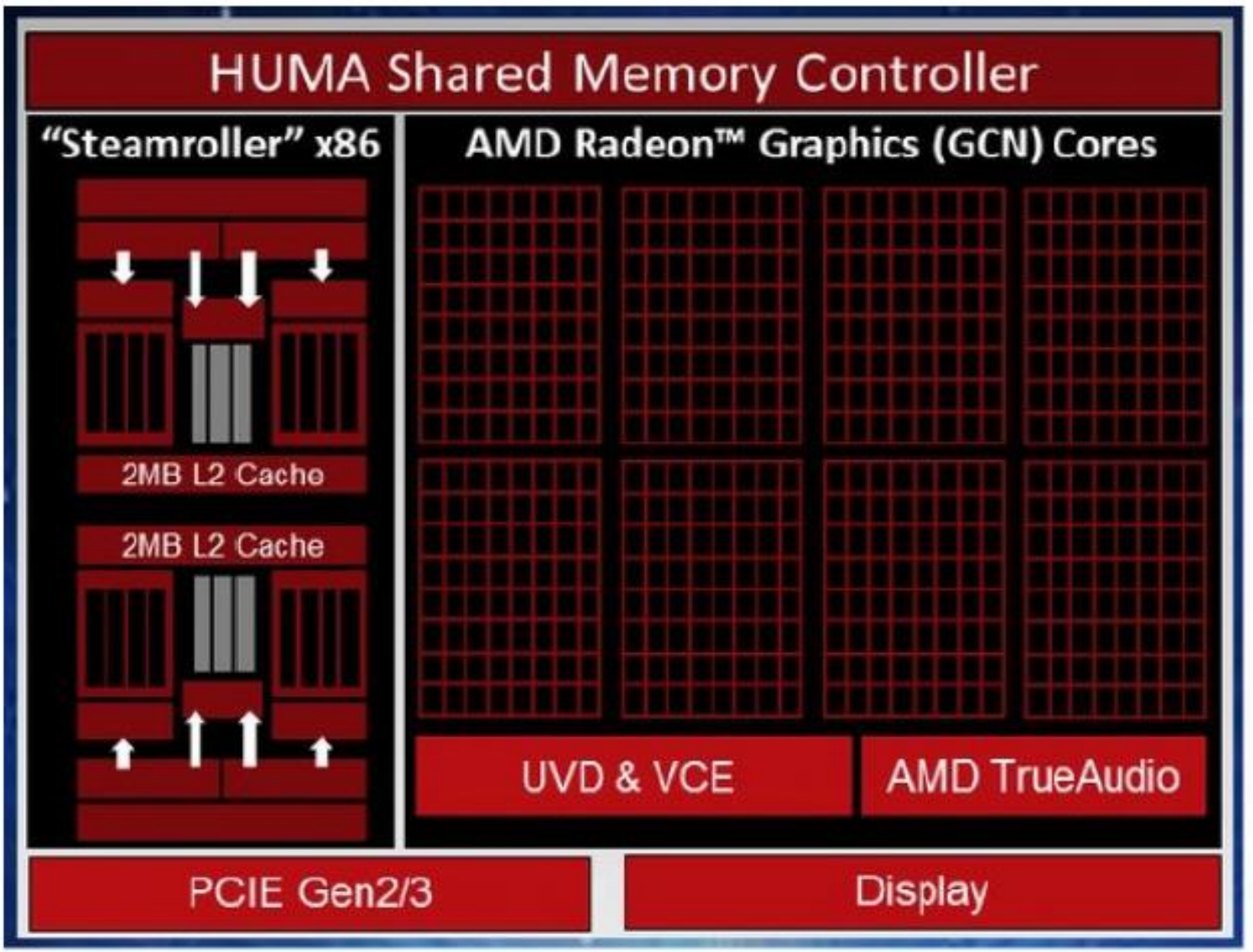
- Empezaron a presentar diseños en los que varios procesadores están implementados sobre un solo chip, dando lugar a la tecnología MULTICORE o MULTINUCLEO

- En una arquitectura MULTICORE cada procesador contiene dos o más nucleos que pueden ejecutar instrucciones de forma simultanea y obtener un "paralelismo perfecto"



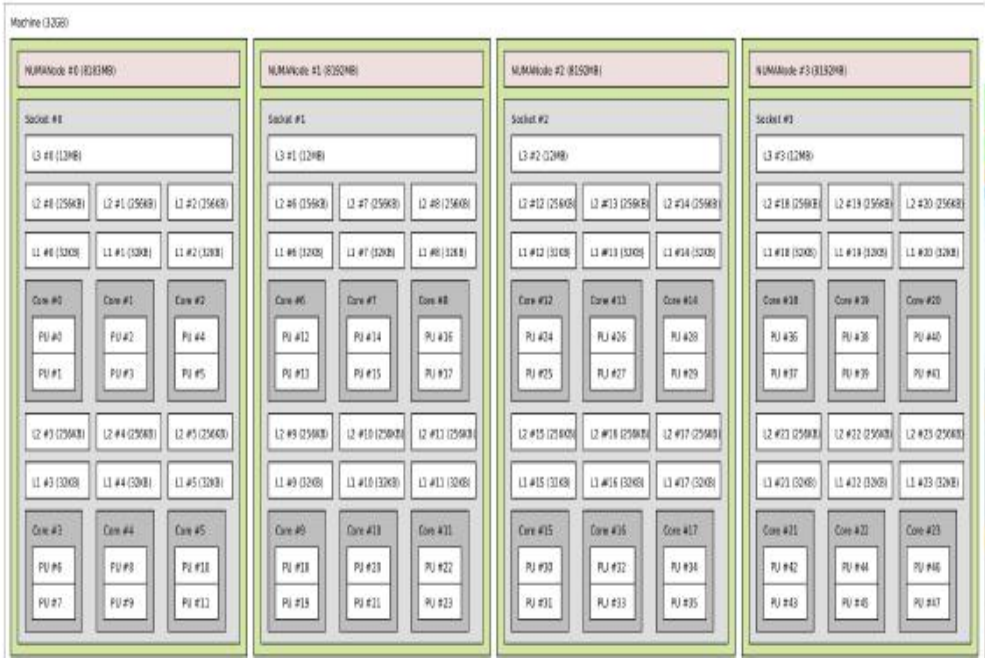

2+2
FDI
PCIE
DMI
DDI
SA
Core
2M
Core
2M
GT-2
DDR3
4+2
FDI
PCIE
DMI
DDI
SA
Core
2M
Core
2M
Core
2M
Core
2M
GT-2
DDR3
ULT 2+3
LPT-LP
OPIO
DDI
OPIO
SA
Core
2M
Core
2M
GT-3
LP DDR3
Intel
4+3
eDRAM
FDI
PCIE
DMI
OPIO
DDI
SA
Core
1.5M
Core
1.5M
Core
1.5M
Core
1.5M
GT-3
DDR3
ULT 2+2
LPT-LP
OPIO
DDI
OPIO
SA
Core
2M
Core
2M
GT-2
LP DDR3

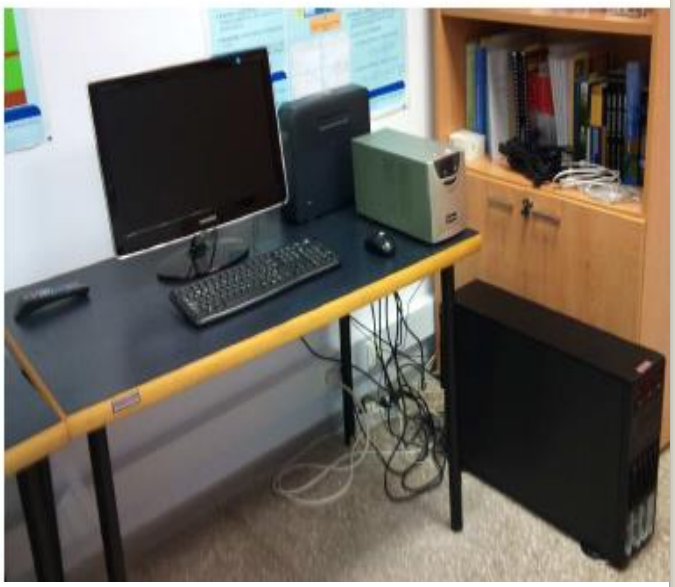
AMD


HUMA Shared Memory Controller
"Steamroller" x86
AMD Radeon™ Graphics (GCN) Cores
2MB L2 Cache
2MB L2 Cache
UVD & VCE
AMD TrueAudio
PCIE Gen2/3
Display

Multicore

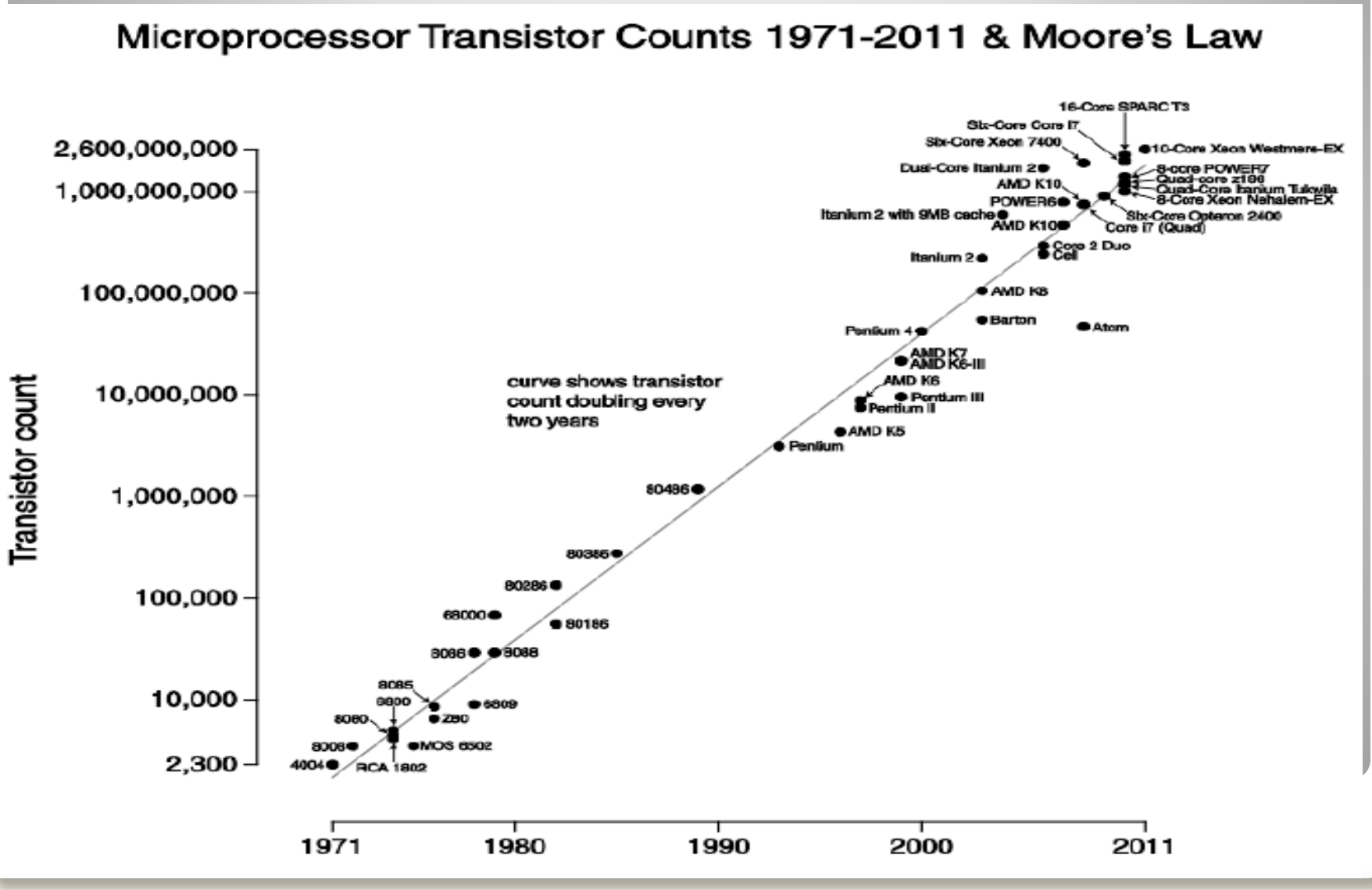
- Los sistemas multicore (multinúcleo) contienen varios cores que tienen acceso a un espacio de memoria común, organizado de forma jerárquica.
- En la actualidad son los sistemas computacionales estándar, y sistemas más complejos se obtienen combinando varios de ellos.
- Se programan a través de hilos (threads), que comparten datos en la memoria común.





LEY DE MOORE

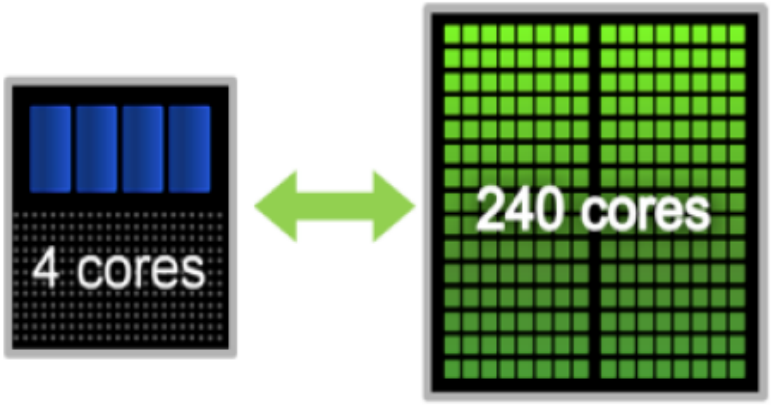
La **ley de Moore** expresa que aproximadamente cada dos años se duplica el número de transistores en un microprocesador.



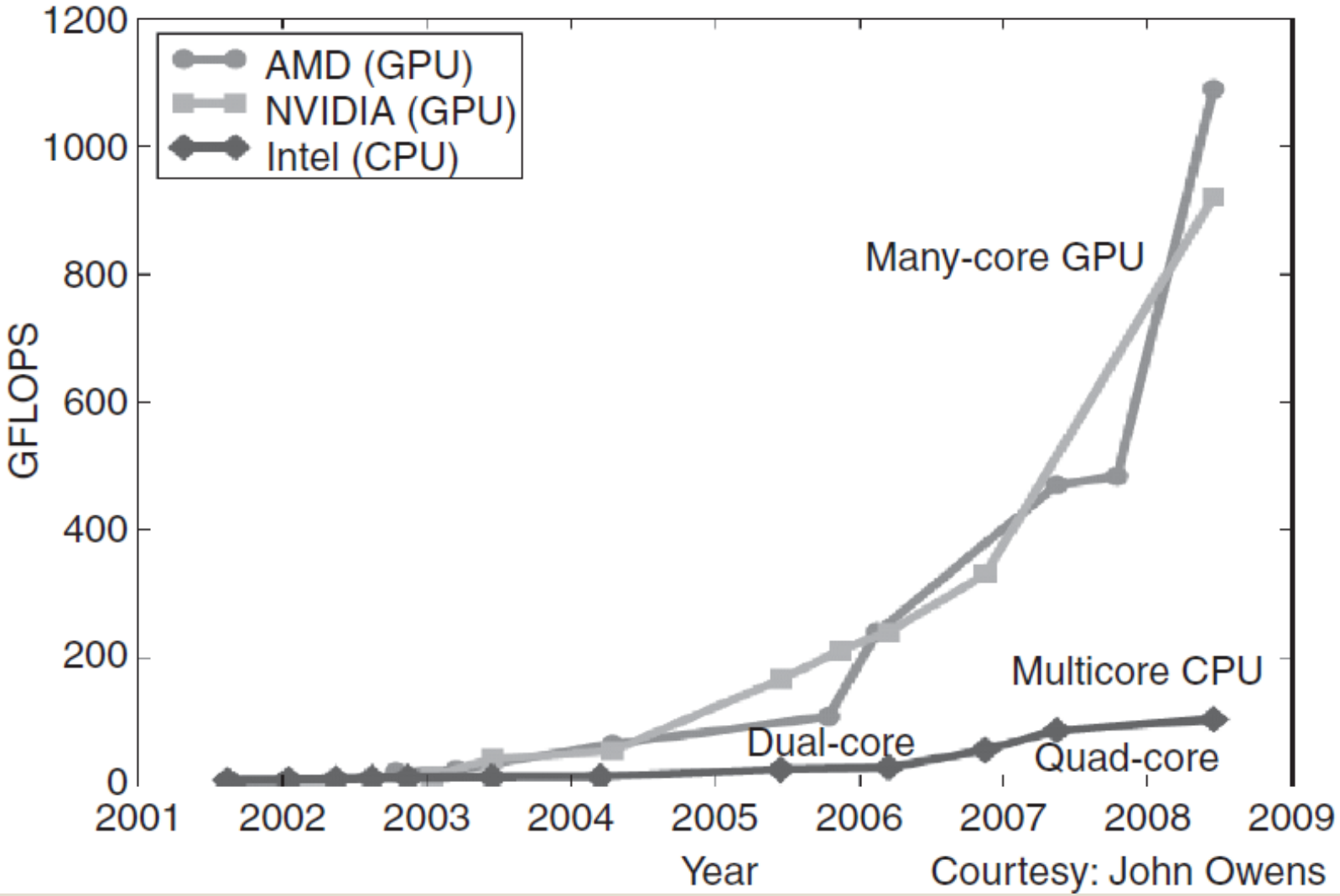
¿Multi-core ó Many-core?

Dos tendencias

- Multi-core
  - Más procesadores en un único componente
- Many-core
  - Muchos (masivamente más) pequeños cores
  - Concentrados en tareas específicas
  - Paralelismo masivo

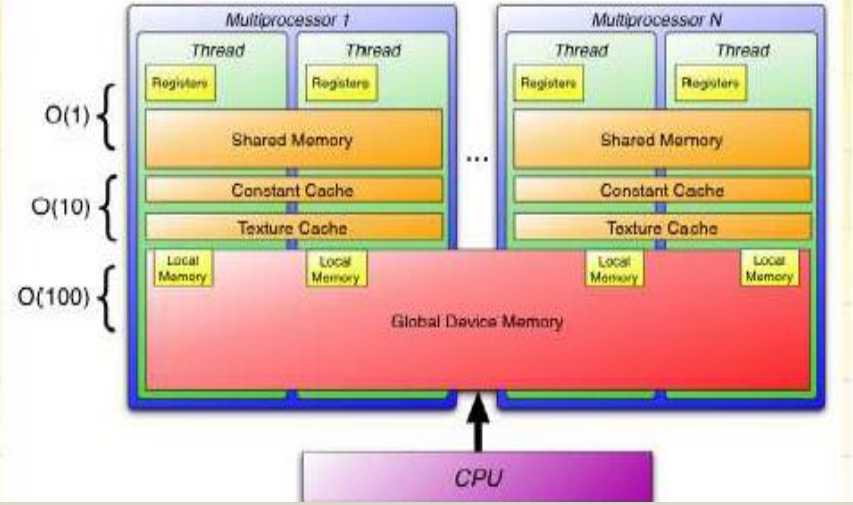


¿Multi-core ó Many-core?



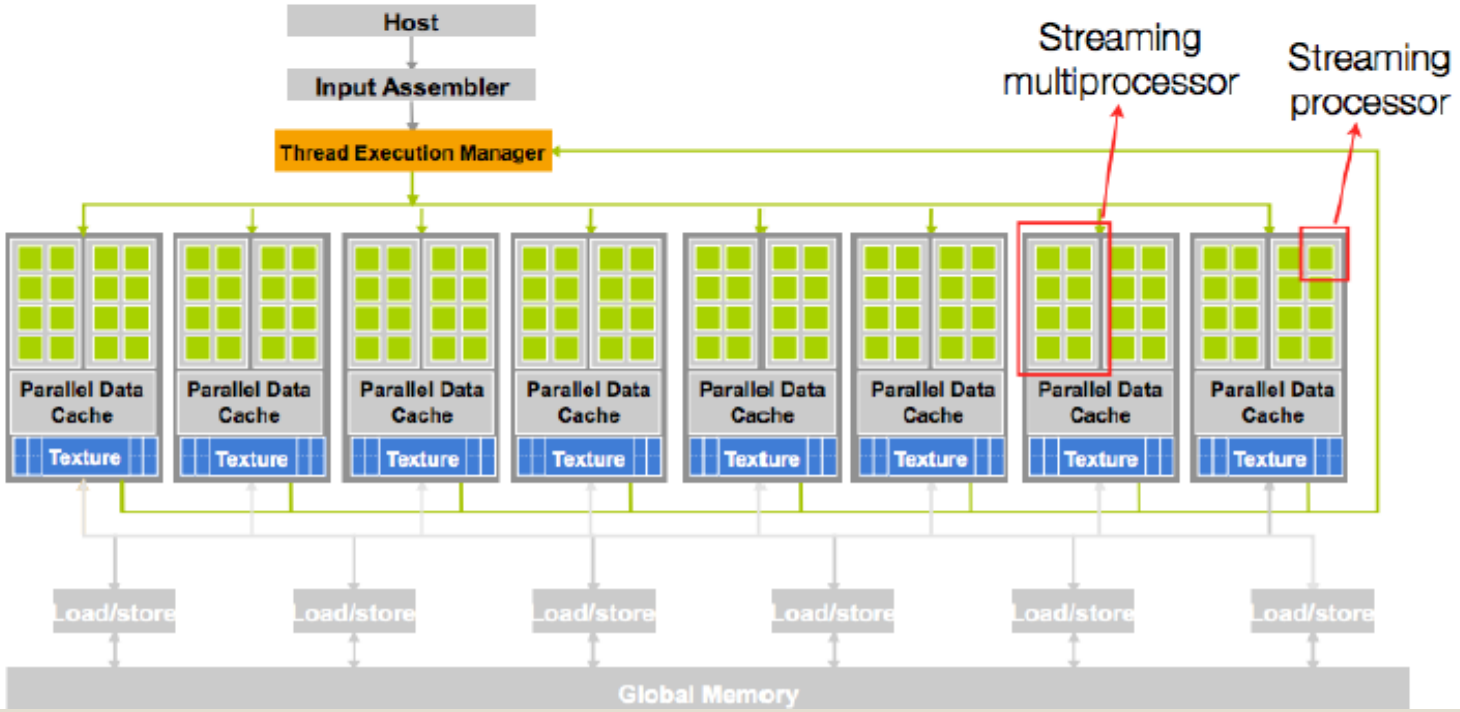
Tarjetas gráficas, GPU

- Originariamente para vídeojuegos, en la actualidad también para procesamiento de propósito general (GPGPU).
- Normalmente como coprocesadores con un sistema multicore, que dispone de una o varias GPUs.
- El programa se ejecuta en CPU y manda trabajos a la GPU.
- Programación dependiente del fabricante (CUDA) y también hay software portable (OpenCL).
- Constan de muchos cores de GPU, organizados en bloques, y con organización jerárquica de la memoria.
- Varios tipos de tarjetas con distintas capacidades computacionales (Gforce, Tesla, Kepler...)



GPU: Graphics Processing Unit

Arquiectura de many-cores



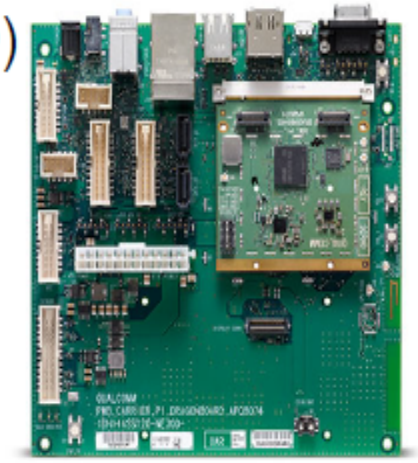
Samsung Exynos 5

- Odroid XU-E and XU3 bareboards
- Sports a Exynos 5 Octa
  - big.LITTLE architecture
  - big: Cortex-A15 quad
  - LITTLE: Cortex-A7 quad
- Exynos 5 Octa 5410
  - Only 4 CPU-cores active at a time
  - GPU: Power VR SGX544MP3 (Imagination Technologies)
    - 3 GPU-cores at 533MHz → 51 GFLOPS
- Exynos 5 Octa 5422
  - All 8 CPU-cores can be working simultaneously
  - GPU: ARM Mali-T628 MP6
    - 6 GPU-cores at 533MHz → 102 GFLOPS

180$

Snapdragon 800

- ***CPU: Quad-core Krait 400 up to 2.26GHz (ARMv7 ISA)***
  - Similar to Cortex-A15. 11 stage integer pipeline with 3-way decode and 4-way out-of-order speculative issue superscalar execution
  - Pipelined VFPv4[2] and 128-bit wide NEON (SIMD)
  - 4 KB + 4 KB direct mapped L0 cache
  - 16 KB + 16 KB 4-way set associative L1 cache
  - 2 MB (quad-core) L2 cache



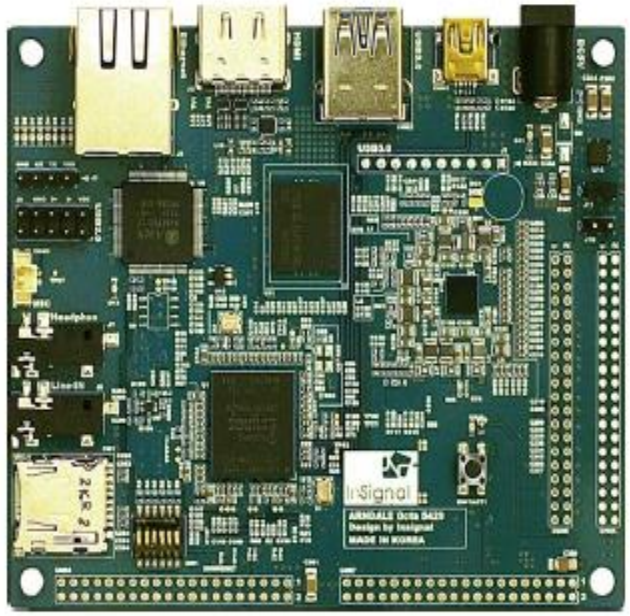
- ***GPU: Adreno 330, 450MHz***
  - OpenGL ES 3.0, DirectX, OpenCL 1.2, RenderScript
  - 32 Execution Units. Each with 2 x SIMD4 units

- ***DSP: Hexagon 600MHz***

More development boards

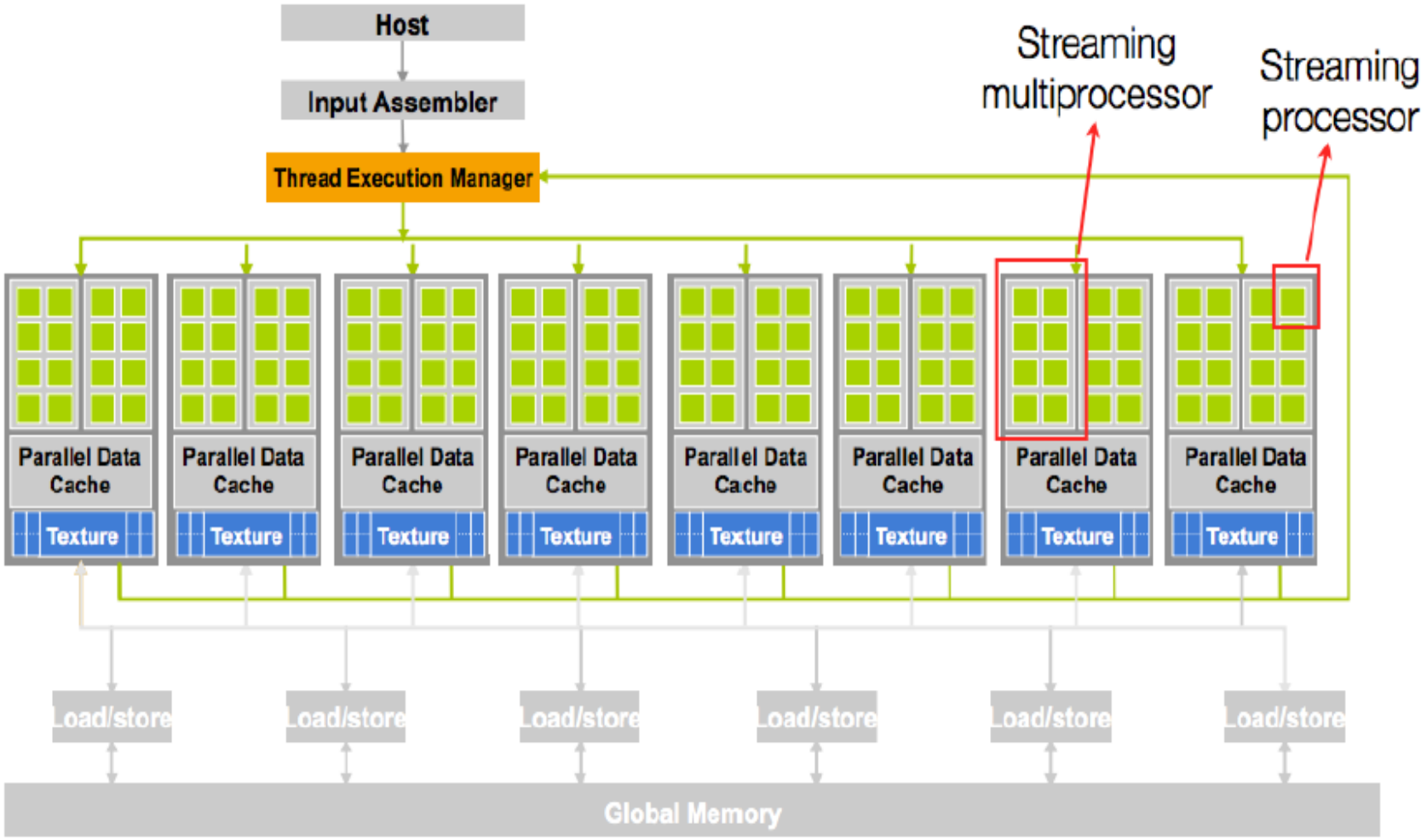
- Jetson TK1 board
  - Tegra K1
  - Kepler GPU with 192 CUDA cores
  - 4-Plus-1 quad-core ARM Cortex A15
  - Linux + CUDA
  - 180$

- Arndale
  - Exynos 5420
  - big.LITTLE (A15 + A7)
  - GPU Mali T628 MP6
  - Linux + OpenCL
  - 200$



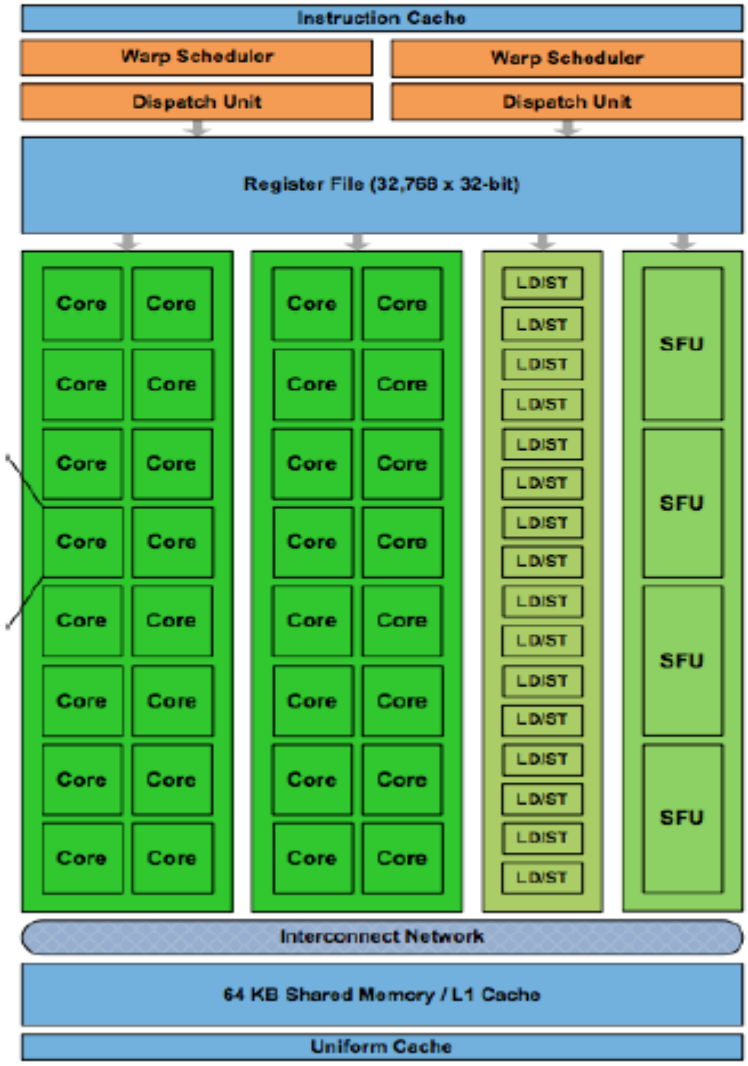
- …

Arquitectura global



Streaming Multiprocessor (SM)

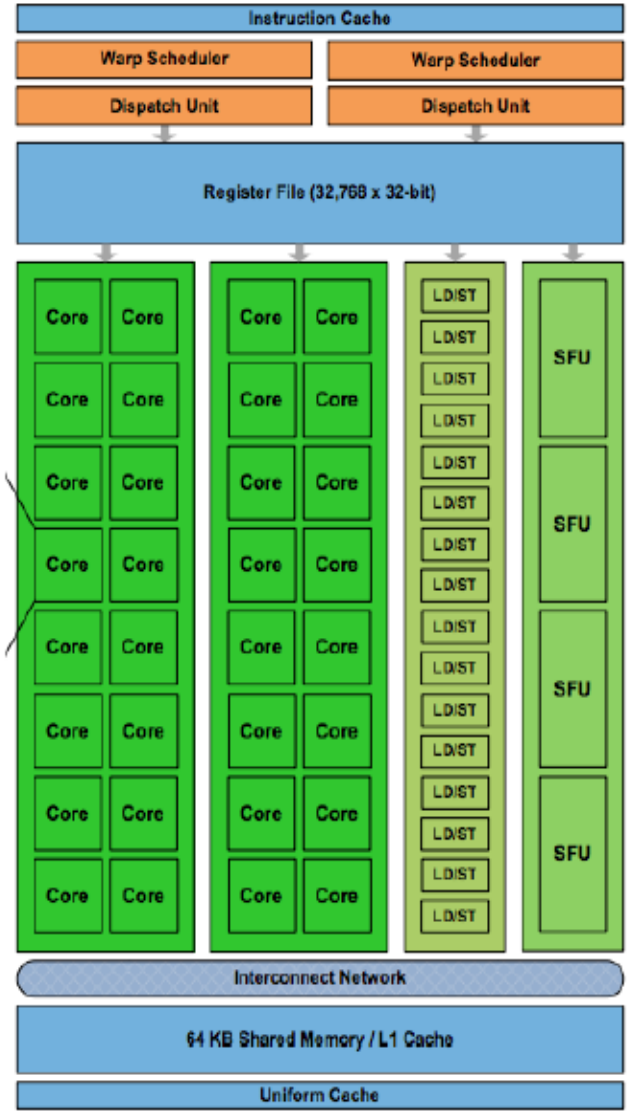
Conjunto de **Streaming Processors (SP)** == CUDA core



Multithreading

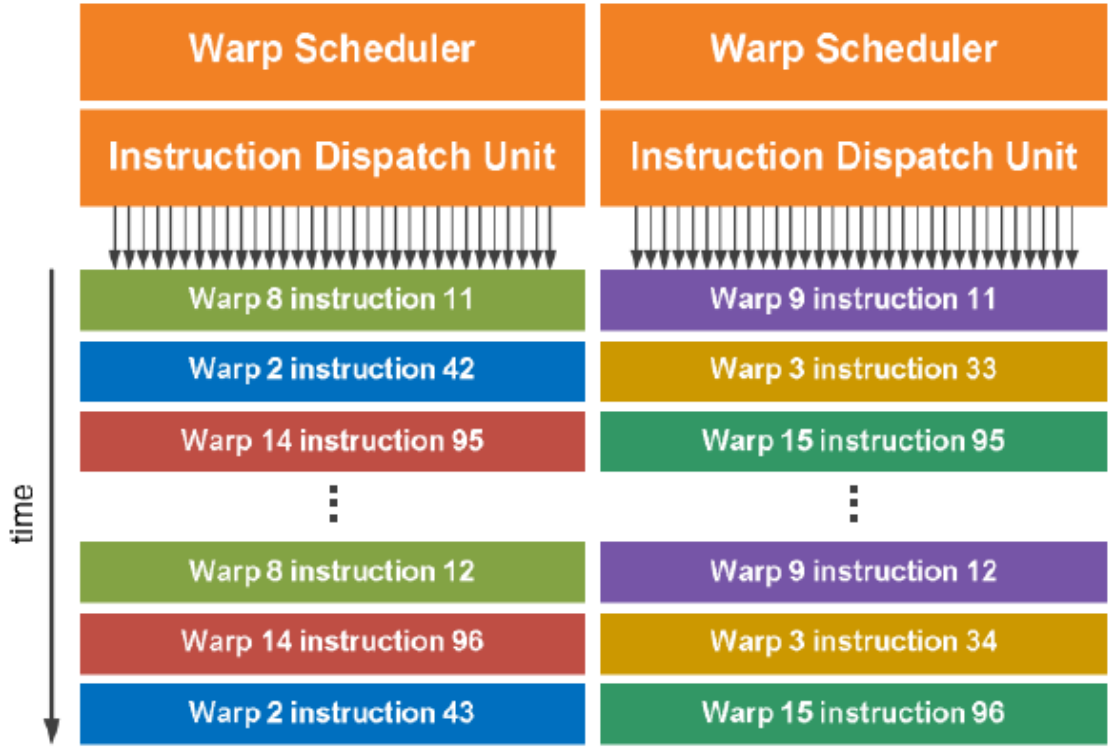
Cada SM es *multithreaded*.

- 768 threads por cada SM
- SM mapea threads a SPs, LD/ST ó SFU
- *Threads* agrupados en *warps* de 32
- SP: CUDA Core
- LD/ST: unidades de carga/almacenamiento
- SFU: Special Function Unit



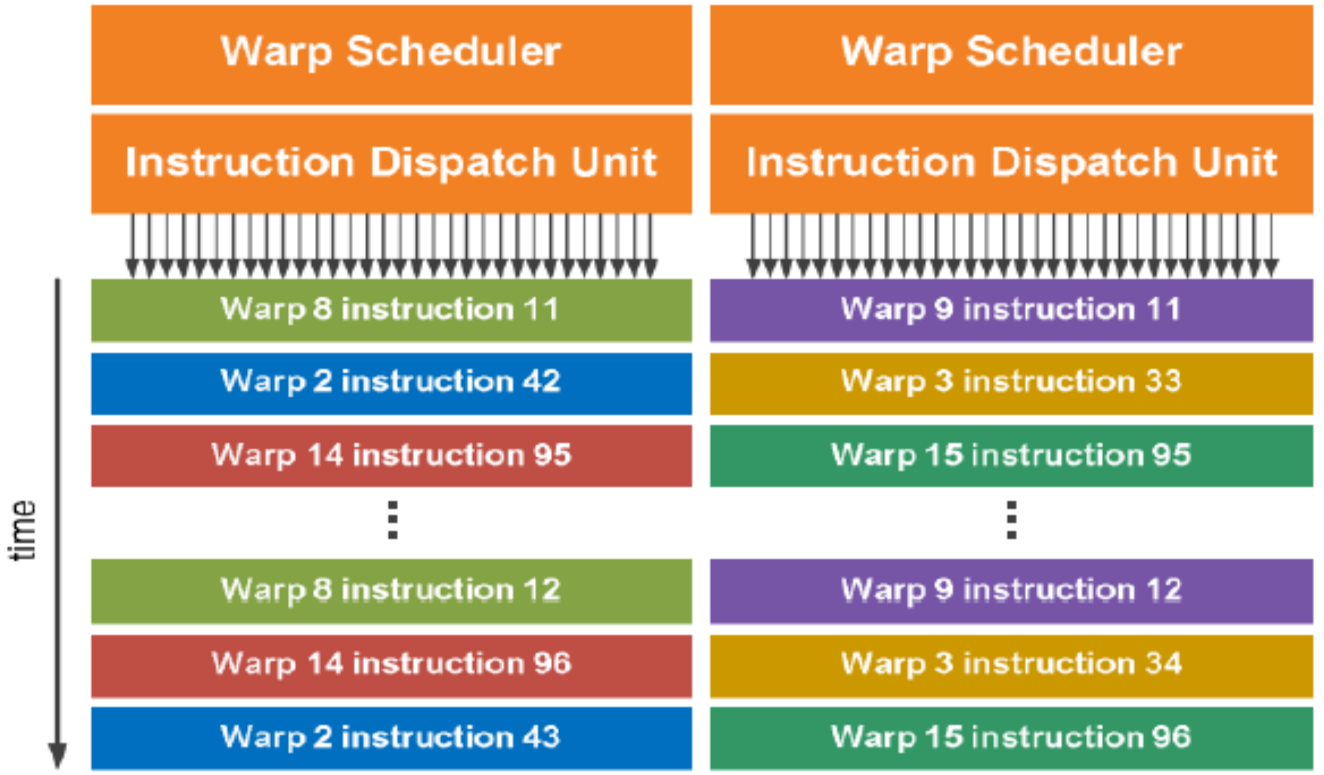
Multithreading

- Cada SM usa dos *warp schedulers*
- Cada SM planifica ejecución de 2 *warps* de 32 *threads* cada uno.
  - *Warps* elegible si todos sus *threads* están en condiciones de ejecutar.
  - Todos los *threads* del *warp* ejecutan la misma instrucción.
  - Simultáneamente en GPU: $2 \times 32 \times 14 = 896$ *threads*



Multithreading

- Cada SM actúa como una máquina SIMD.
- Conjunto de 24 *warps*, cada uno con 32 *threads*
  - Cada SM puede planificar hasta $24 \times 32 = 768$ *threads*
  - GPU puede contener $768 \times 14 = 10752$ *threads*
- Latencia de memoria $\sim$ 400-800 ciclos.



Compute Unified Device Architecture

Modelo de programación desarrollado por NVIDIA

- De propósito general, pero especializado para GPGPU
- API para cómputo paralelo
- API para administración de memoria

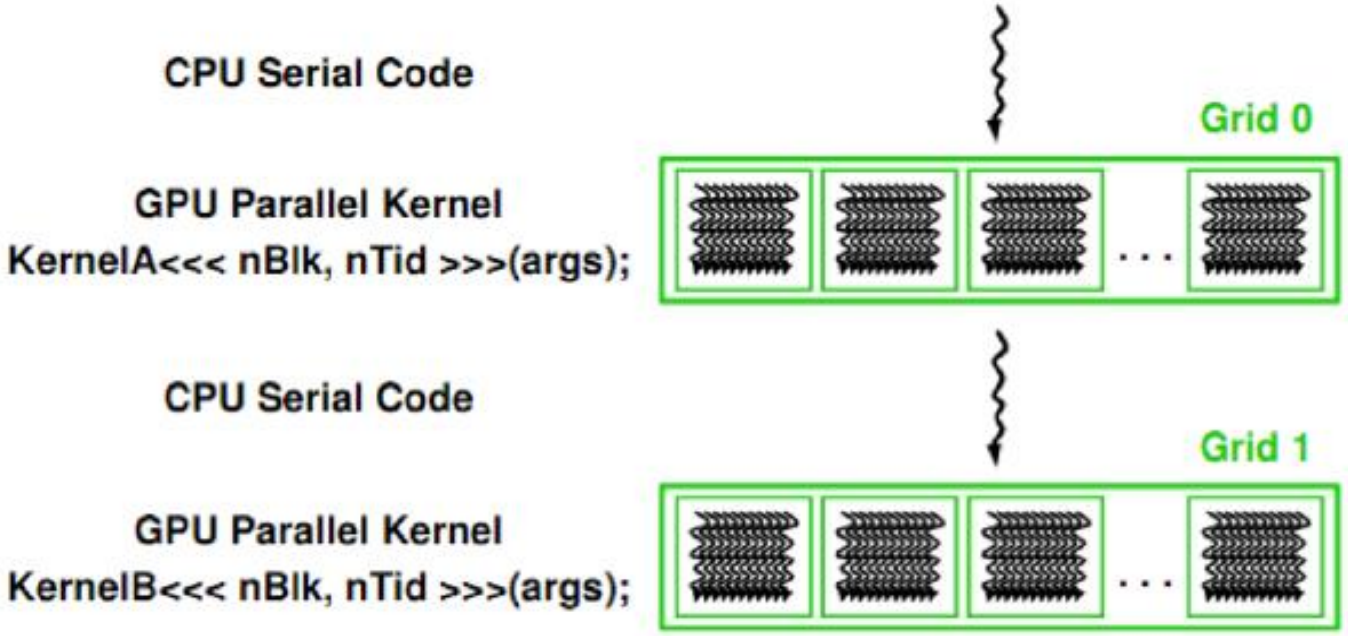
Computación puede ocurrir en dos lugares

- **Host**. CPU. Código escrito en C/C++
- **Device**. GPU. Código escrito en CUDA C
  - C con extensiones
- *Host* y *device* tienen memorias separadas
- Programa CUDA contiene código para *host* y para *device*

CUDA

Estructura general de un programa CUDA

- Código serial se ejecuta en CPU
- Código paralelo se ejecuta en GPU



CUDA Kernel

¿Kernel?

- Código paralelo que puede ser ejecutado por múltiples *threads* en *device*
- Es llamado desde el *host*
- Todos los *threads* ejecutan el mismo código

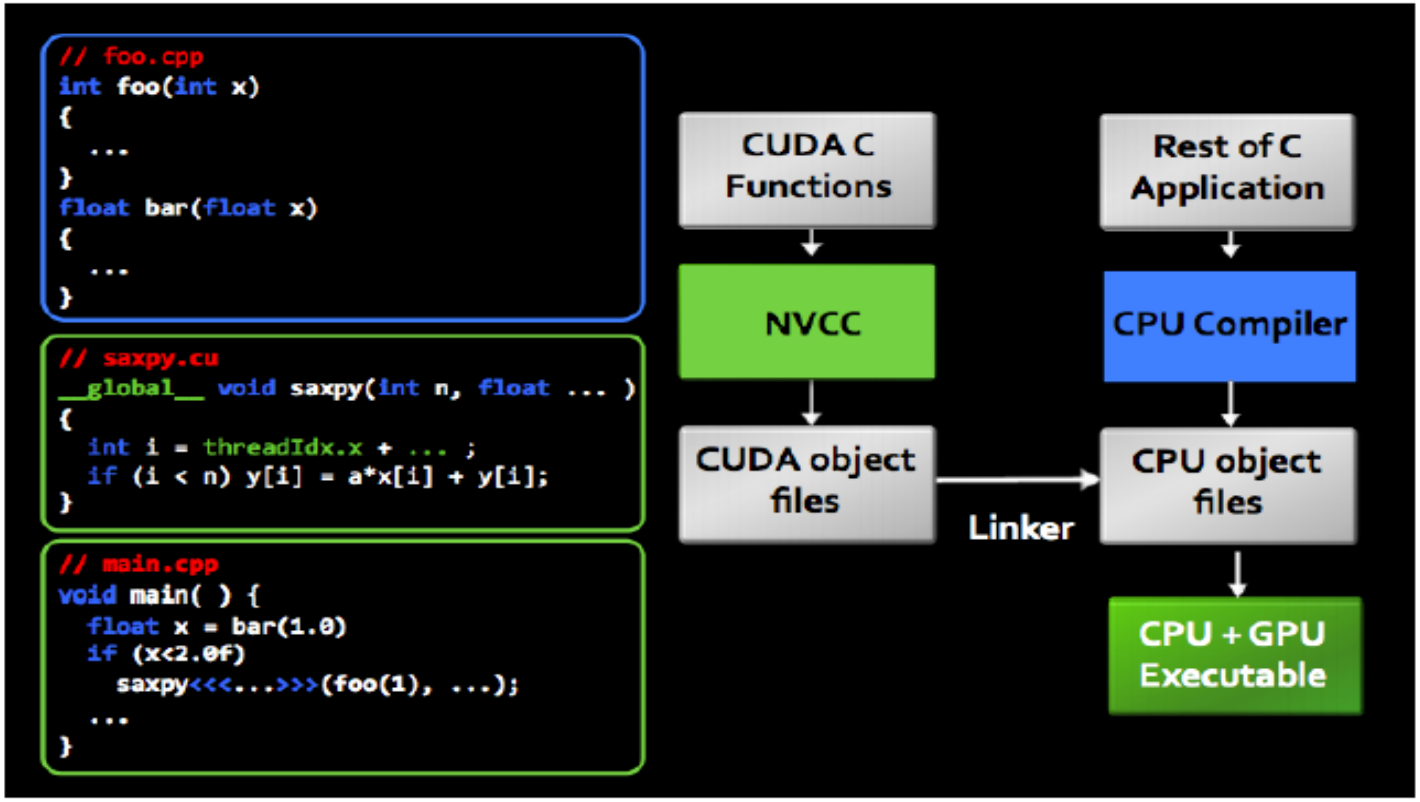
¿Cómo saber dónde se ejecuta?

- Calificadores de funciones
  - `__global__` kernel GPU, lanzado desde CPU. Debe retornar `void`.
  - `__device__` sólo puede ser llamado desde GPU.
  - `__host__` sólo puede ser llamado desde CPU.
  - `__host__` pueden ser simultáneos `__device__`

Compilando CUDA

Compilador `nvcc`

- Compila código para *device*
- Código "normal" se compila con GCC



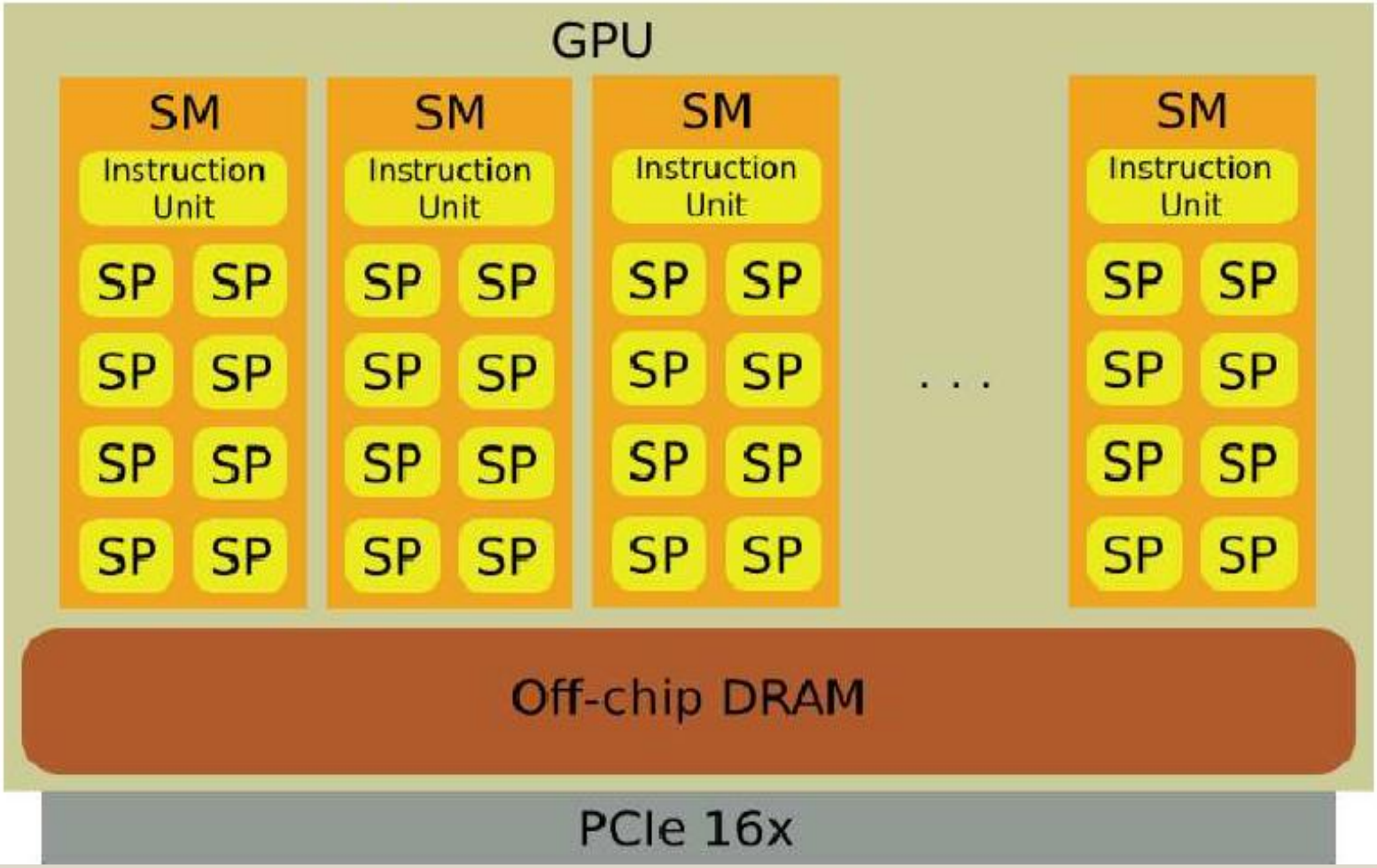
SIMD- Sistemas

- Modelo SIMD: muchas unidades de proceso, cada una realizando la misma operación (SI) y cada una sobre sus datos (MD).
- En la actualidad podemos considerar sistemas SIMD coprocesadores como:
  - tarjetas gráficas (GPU), hasta ≈ 2500 cores. En los sistemas para gráficos, se pueden programar para propósito general.
  - Intel Xeon Phi, entre 57 y 61 cores, cada uno hasta 4 threads por hardware.

pero también hacen computación de forma asíncrona, normalmente trabajo en CPU y se manda parte del procesamiento al coprocesador.

GPU- Estructura

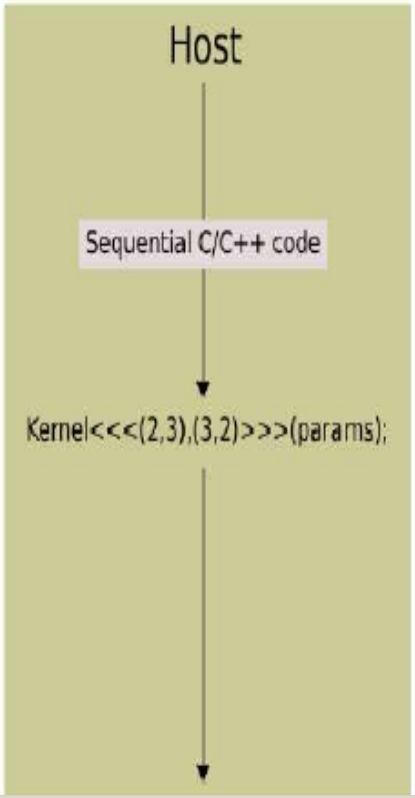
Constan de varios Streaming Multiprocessors (SMs), cada uno con varios Streaming Processors (SPs):

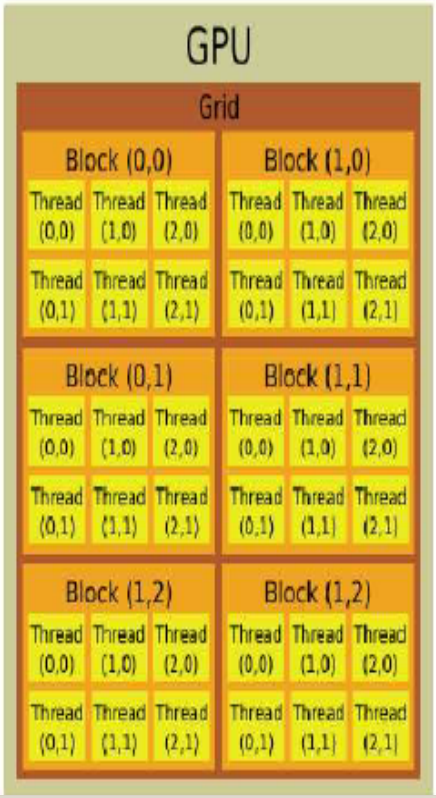


GPU- Modelo programación

- Los procesadores (SPs) de un SM ejecutan hilos independientes, pero en cada instante ejecutan la instrucción leída por la Instruction Unit (IU).
- En cada SM los hilos los gestiona el hardware: bajo coste.
- kernel es la parte de código en la CPU que lanza ejecución a GPU:

- Descompone un problema en subproblemas y lo mapea sobre un grid, que es un vector 1D o 2D de bloques de hilos.
- Cada bloque es un vector 1D, 2D o 3D de hilos.
- Los hilos usan su identificador de thread dentro de un bloque y de bloque dentro del grid para determinar el trabajo que tienen que hacer.





Entornos de programación de sistemas SIMD

- Programación de GPU es programación específica:
  - CUDA para tarjetas NVIDIA
  - OpenCL es estándar para tarjetas de diferentes fabricantes.
- En Intel Xeon Phi se puede usar OpenMP y MPI,
  con compilando diferenciada para CPU y para el Xeon Phi,
  pero
  pudiendo trabajar de forma conjunta.

Sistemas paralelos

Sistema	Cores	Programación
portátil	2-4	OpenMP
PC	2-8	OpenMP
servidor	4-24	OpenMP
NUMA (Ben)	16-256	OpenMP
cluster	2-32 $\times$ 2-8	MPI+OpenMP
supercomputador (Arabí)	32-1024 $\times$ 8-32	MPI+OpenMP
GPU	112-512	CUDA
heterogéneo	cluster+NUMA	MPI+OpenMP
jerárquico	cluster+NUMA+GPU	MPI+OpenMP+CUDA
$\vdots$		

- PROGRAMACIÓN MULTICORE EN JAVA
- 1. Se identifican cuantos núcleos tiene nuestro procesador:
- `int processors = Runtime.getRuntime().availableProcessors();`
- 2. Se crean los procesos independientes
- `ExecutorService executor = Executors.newFixedThreadPool(processors);`
- ExecutorService es una interface que hereda de Executor, como se puede apreciar se crea un pool de Threads y como parametros le mandamos el numero de procesadores que tiene el cliente donde se ejecutara.

- PROGRAMACIÓN MULTICORE EN JAVA
- 3. Se lanzan los procesos a ejecución
- ```
  executor.execute(new Runnable(){
  public void run(){
  //aqui se hace algo ...
  }
  });
  ```
  ```

- PROGRAMACIÓN MULTICORE EN JAVA
- 4. Se da de baja el ejecutor
- `executor.shutdown();`