

# JORNADA DE DIVULGACIÓN DE APLICACIONES CIENTÍFICAS SOBRE PROCESADORES GRÁFICOS Y QUANTUM COMPUTING

**JGPUQC 2023 – UNIVERSIDAD DE ALICANTE**

Manuel Benavent-Lledó <[mbenavent@dtic.ua.es](mailto:mbenavent@dtic.ua.es)>

David Mulero-Pérez <[dmulero@dtic.ua.es](mailto:dmulero@dtic.ua.es)>

José García-Rodríguez <[jgarcia@dtic.ua.es](mailto:jgarcia@dtic.ua.es)>

# DAVID MULERO

## PÉREZ

- **Grado en Ingeniería Multimedia**
  - Universidad de Alicante
  - 2017 – 2021
- **Máster en Ciencia de Datos**
  - Universidad de Alicante
  - 2021 – 2022
- **Doctorado en Informática (Virtual Reality and Deep Learning)**
  - Universidad de Alicante
  - 2022 – Actualidad

MAIL: **DMULERO @ DTIC.UA.ES**

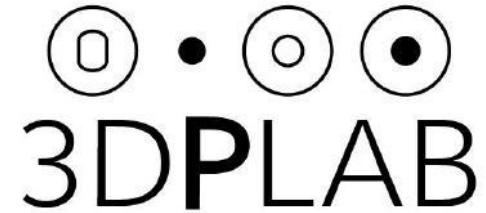
# MANUEL BENAVENT

LLEDÓ

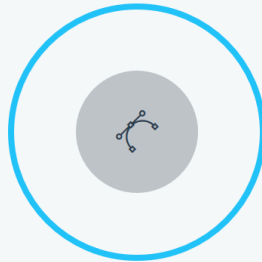
- **Grado en Ingeniería Informática**
  - Universidad de Alicante
  - 2017 – 2021
- **Máster en Automática y Robótica**
  - Universidad de Alicante
  - 2021 – 2022
- **Doctorado en Informática (Deep Learning and Computer Vision)**
  - Univerisdad de Alicante
  - 2022 – Actualidad

MAIL: **DMULERO @ DTIC.UA.ES**

# DEDICACIÓN INVESTIGACIÓN

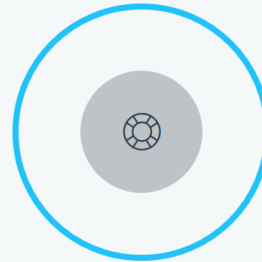


The 3D Perception Lab at the University of Alicante is a group of researchers interested in the intersection of machine learning and computer vision. Our research mission focuses on various aspects of perception often related with mobile robotics in which we exploit 3D data as the main source of information. Some of our research lines include object recognition, semantic segmentation, rigid and non-rigid registration, visual localization and mapping, behavior analysis, and depth estimation. Apart from those general lines we are also highly interested in making those solutions run efficiently by leveraging GPU acceleration using CUDA. Aside from 3D data as our backbone, we are also tied together by our shared vision in the great potential of artificial intelligence, mainly deep learning, which we try to apply and push its limits in every project we work on.



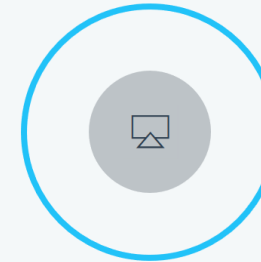
## DEEP LEARNING

Artificial intelligence applied to computer vision and robotics (semantic segmentation, depth estimation, scene understanding, object recognition, localization and mapping).



## 3D COMPUTER VISION

Traditional computer vision methods and new challenges for 3D data (rigid registration, non-rigid registration, reconstruction).



## GPU COMPUTING

Acceleration of computer vision methods and artificial intelligence pipelines for real-time execution and maximum efficiency.

# ESTRUCTURA

## PROCESAMIENTO PARALELO CON GPUS [2h]

### TEORÍA

HISTORIA DE LA GPGPU  
ARQUITECTURA CUDA

### PRÁCTICA

SUMA DE VECTORES



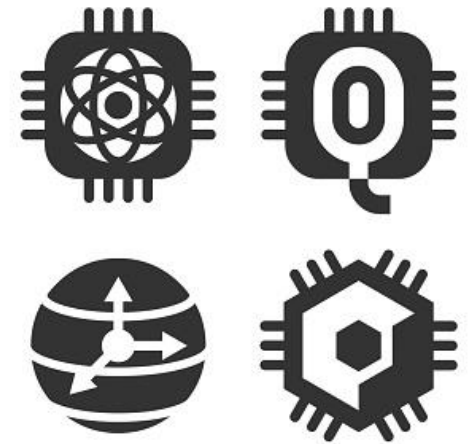
## COMPUTACIÓN CUÁNTICA [2h]

### TEORÍA

COMPUTACIÓN CUÁNTICA  
BASES MATEMÁTICAS

### PRÁCTICA

PROGRAMACIÓN CUÁNTICA



# INICIOS Y EVOLUCIÓN DE LOS PROCESADORES GRÁFICOS (GPUs)

JORNADA DE DIVULGACIÓN DE APLICACIONES CIENTÍFICAS SOBRE  
PROCESADORES GRÁFICOS Y QUANTUM COMPUTING

JGPUQC 2023 - UNIVERSIDAD DE ALICANTE

Manuel Benavent-Lledó <[mбенавент@dtic.ua.es](mailto:mбенавент@dtic.ua.es)>

David Mulero-Pérez <[dmulero@dtic.ua.es](mailto:dmulero@dtic.ua.es)>

José García-Rodríguez <[jgarcia@dtic.ua.es](mailto:jgarcia@dtic.ua.es)>

# CONTENIDO

**La Ley de Moore**

**La Unidad de Procesamiento Gráfico (GPU)**

**Primeros Pasos en Computación sobre GPUs**

**La Arquitectura CUDA**

**¿Qué es CUDA?**

**Arquitectura Hardware**

**Arquitectura Software**

# LA LEY DE MOORE

GORDON MOORE

**Cramming More Components onto Integrated Circuits.** Gordon E. Moore, 1965

**“ THE NUMBER OF TRANSISTORS ON A CHIP DOUBLES EVERY 12 MONTHS”**

– GORDON MOORE, COFUNDADOR DE INTEL, 1965

**“ THE NUMBER OF TRANSISTORS ON A CHIP DOUBLES EVERY 24 MONTHS”**

– GORDON MOORE, COFUNDADOR DE INTEL, 1975





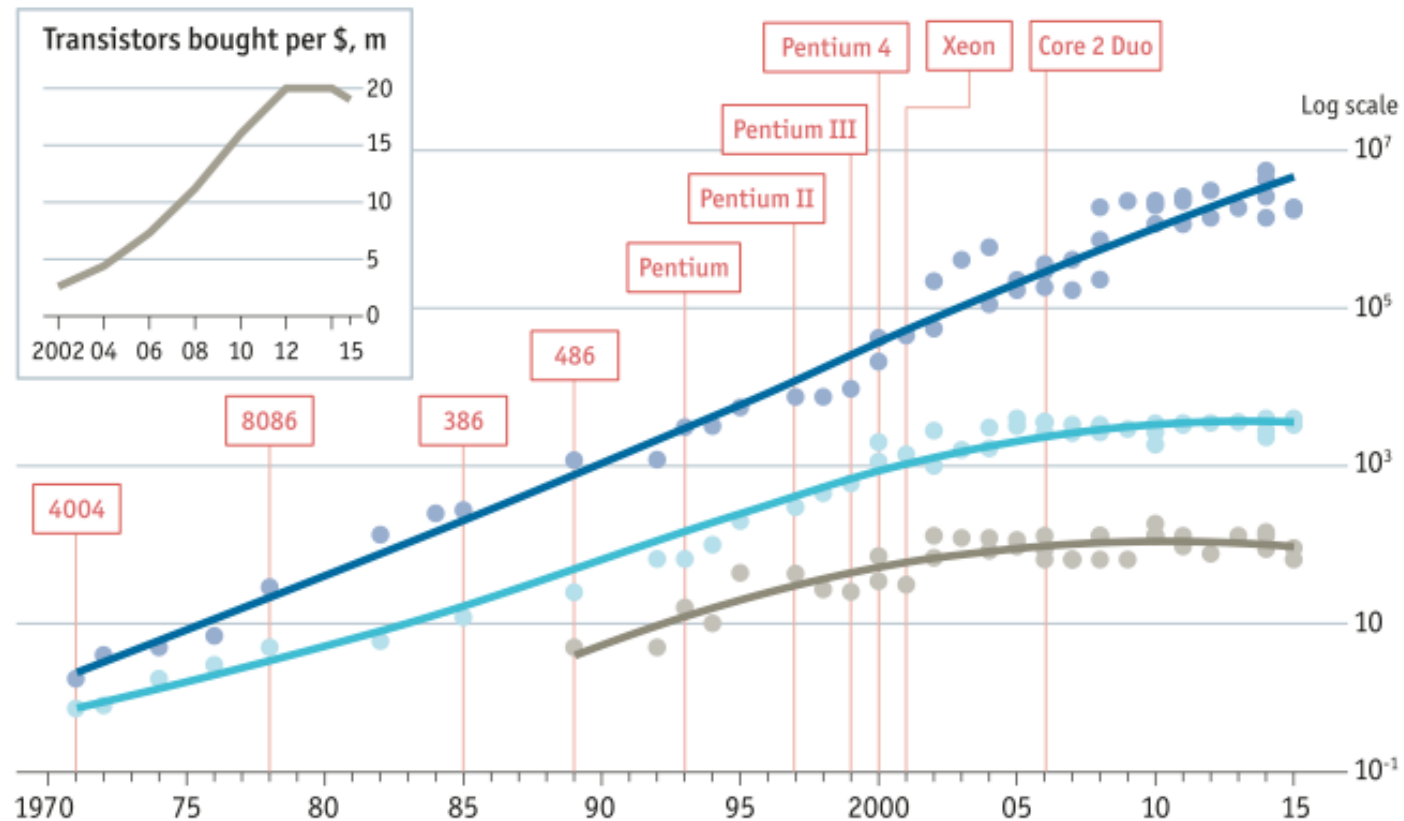
# LA LEY DE MOORE

CADA DOS AÑOS, APROXIMADAMENTE, SE DUPLICA EL NÚMERO DE TRANSISTORES

## Stuttering

● Transistors per chip, '000 ● Clock speed (max), MHz ● Thermal design power\*, w

Chip introduction dates, selected

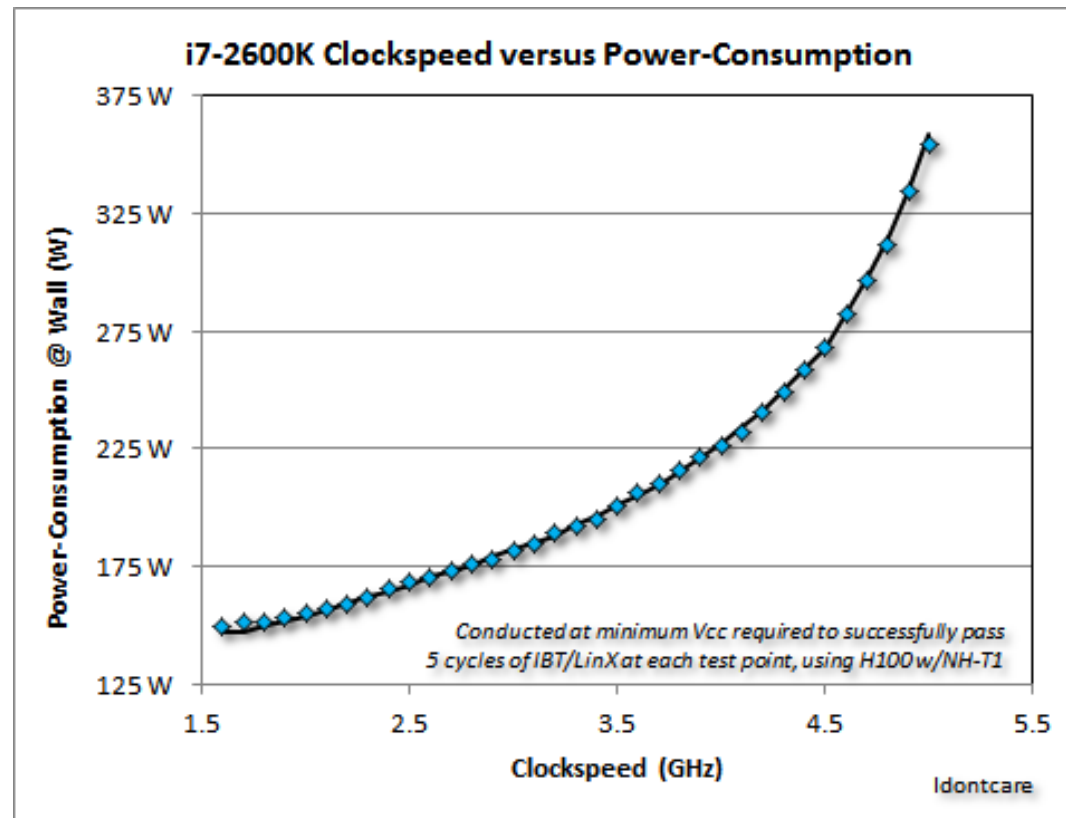


Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; *The Economist*

\*Maximum safe power consumption

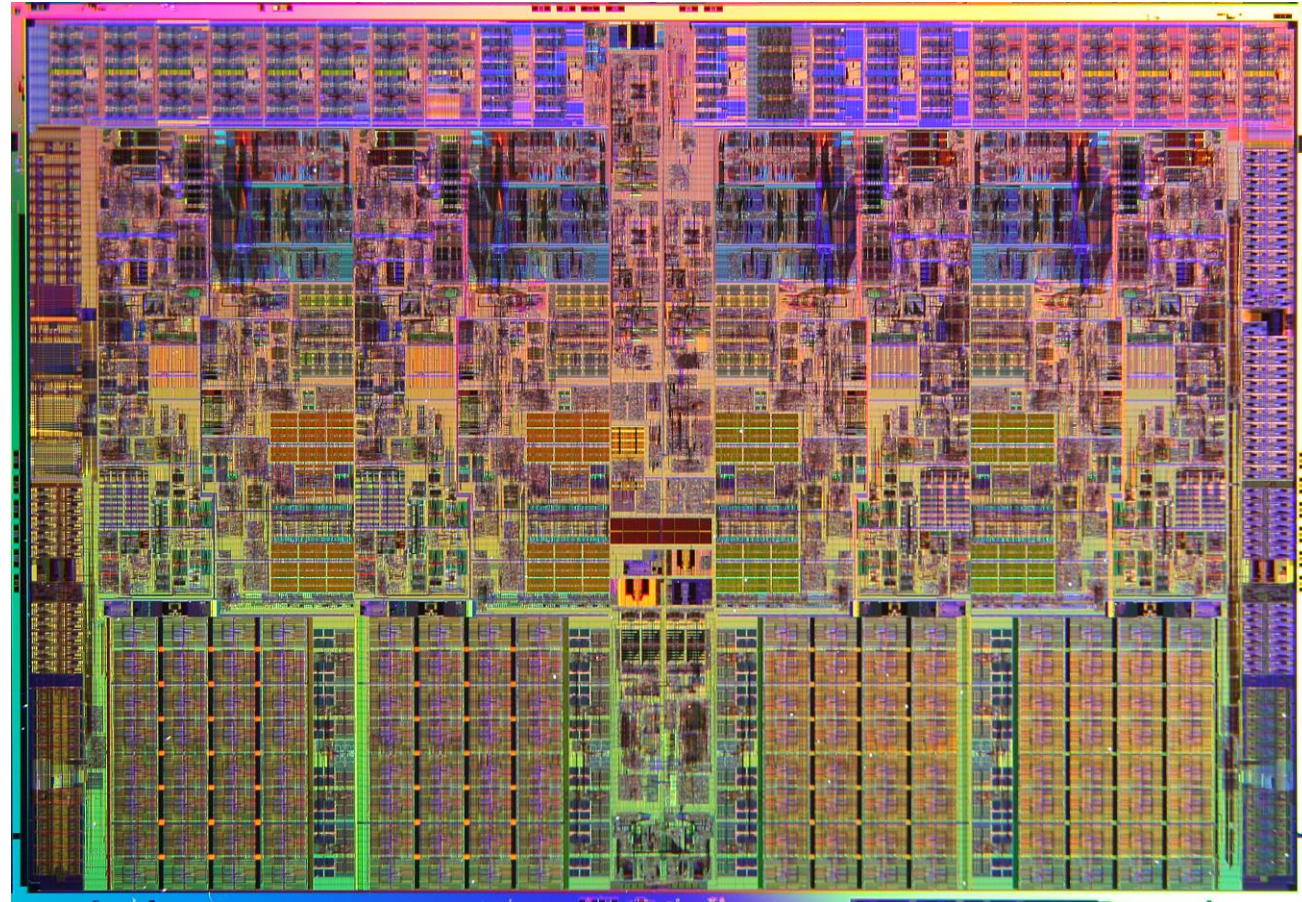
# LA LEY DE MOORE

PROBLEMAS: TAMAÑO DEL TRANSISTOR Y EFECTO TÚNEL  
CONSUMO ENERGÉTICO Y DISIPACIÓN DE CALOR



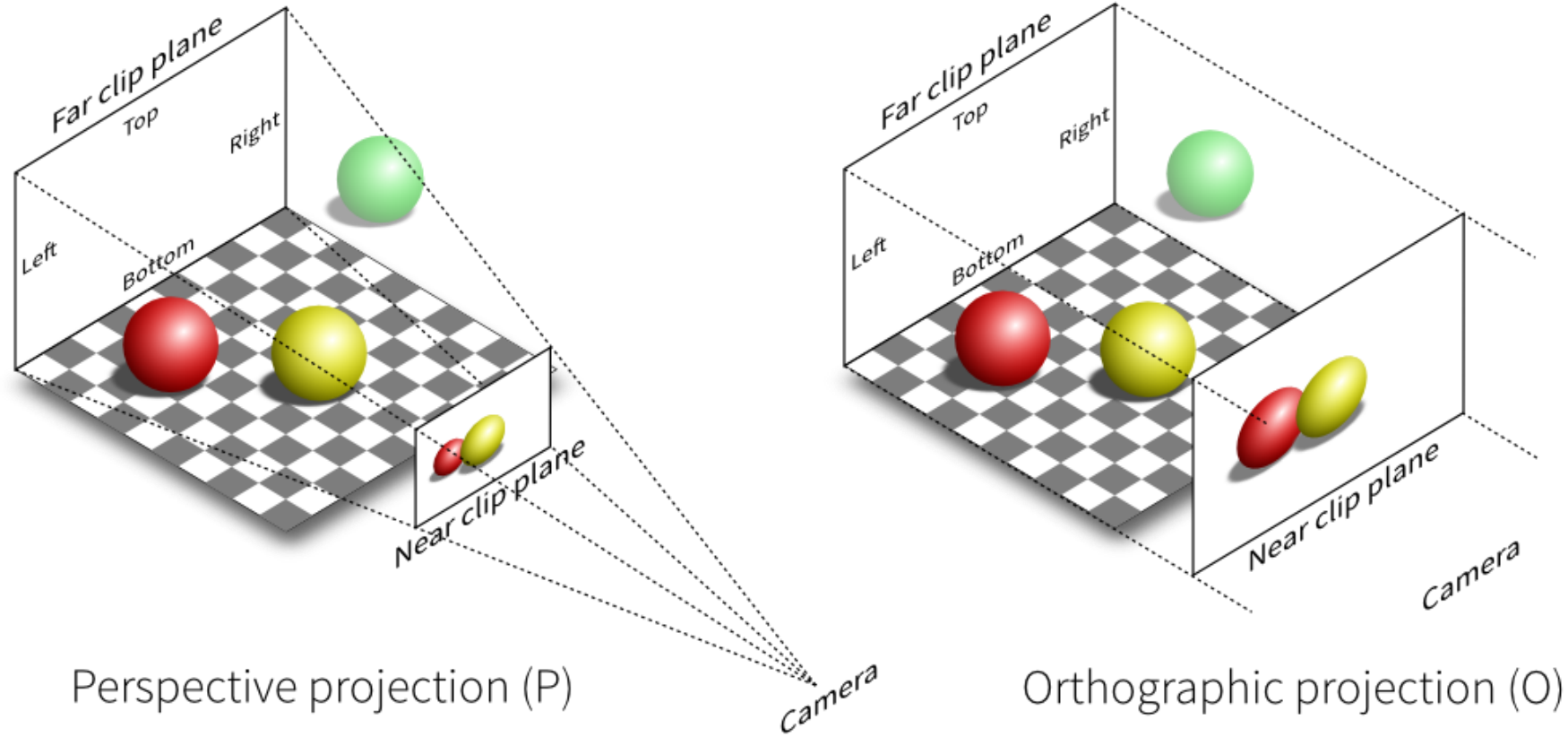
# LA LEY DE MOORE

¿CÓMO CONTINUAR ESCALANDO? MULTICORE



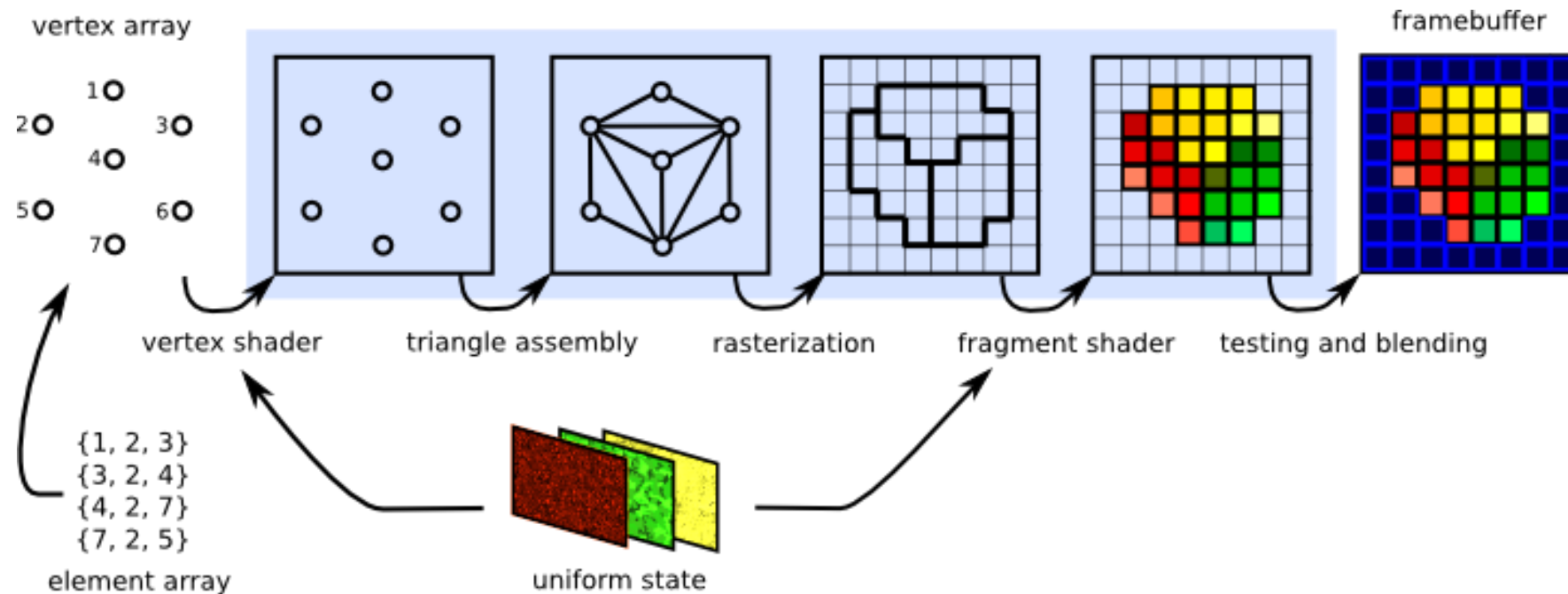
# LA UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)

## EL PIPELINE GRÁFICO: RENDERING



# LA UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)

## EL PIPELINE GRÁFICO: RENDERING





# LA UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)

GEFORCE 256 (1999)



# LA UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)

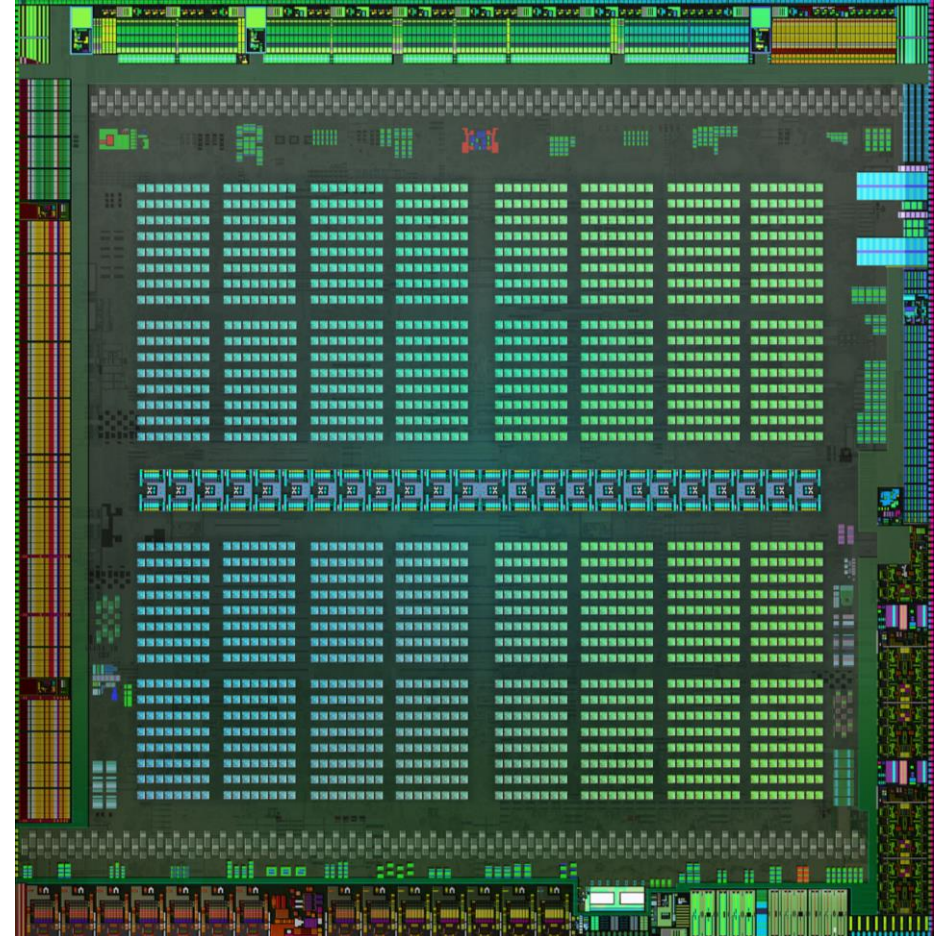
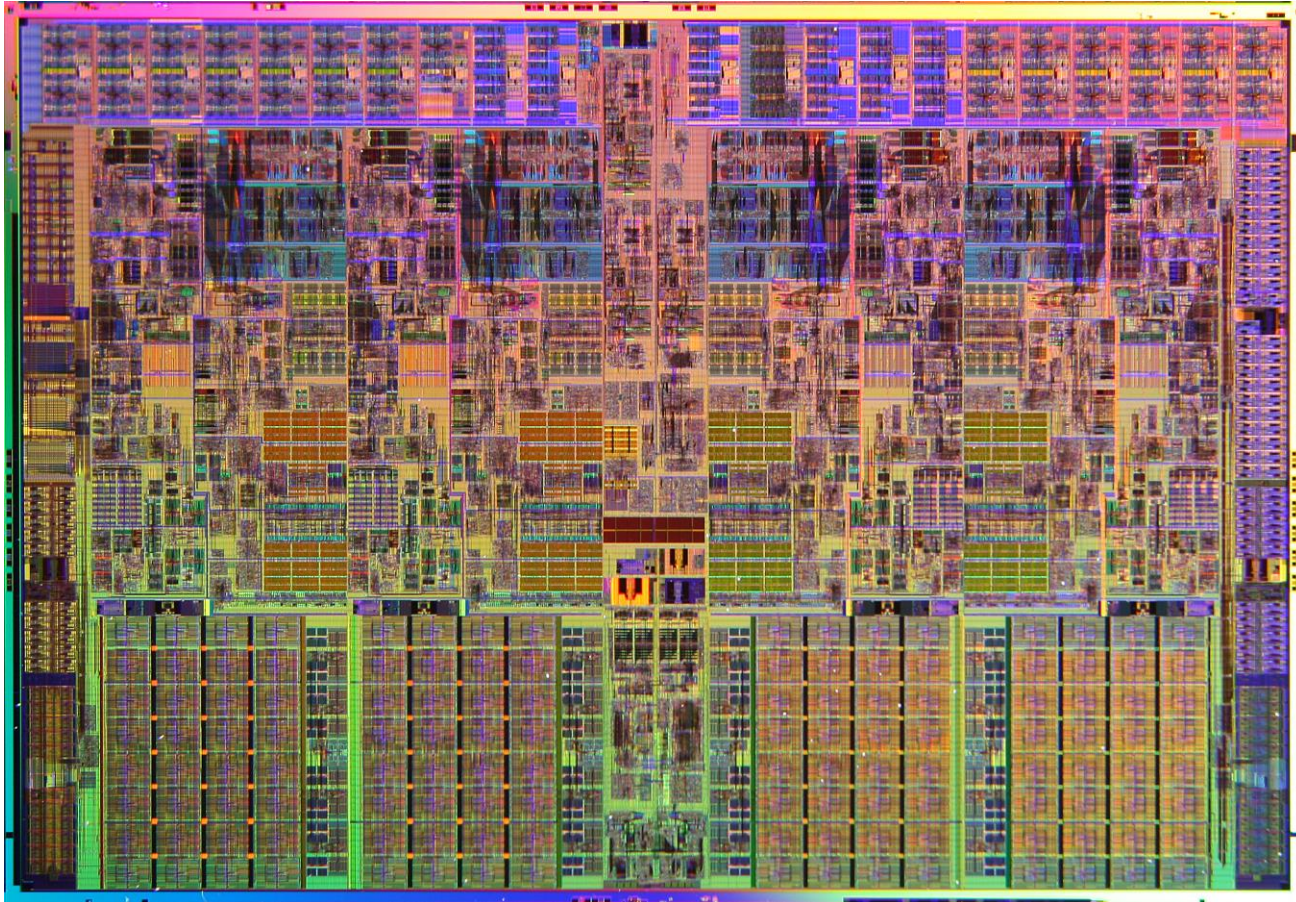
## GEFORCE 256 (1999)

"A SINGLE-CHIP PROCESSOR WITH INTEGRATED TRANSFORM, LIGHTING, TRIANGLE SETUP/CLIPPING, AND RENDERING ENGINES THAT IS CAPABLE OF PROCESSING A MINIMUM OF 10 MILLION POLYGONS PER SECOND."



# LA UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)

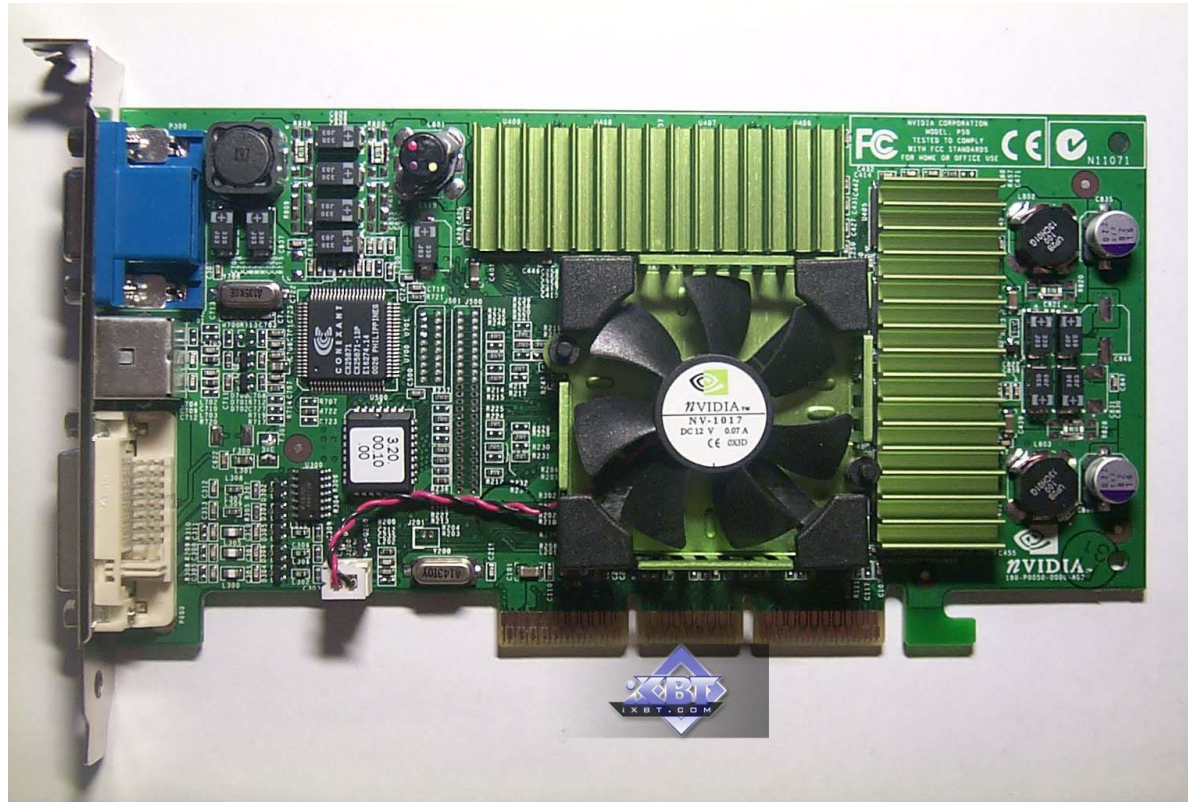
## LA ESENCIA DE LA GPU





# PRIMEROS PASOS EN COMPUTACIÓN SOBRE GPU<sub>s</sub>

## GEFORCE 3 CON VERTEX Y PIXEL SHADERS PROGRAMABLES (2001)



# PRIMEROS PASOS EN COMPUTACIÓN SOBRE GPUS

## ENGAÑAR A LA GPU EMPLEANDO APIs GRÁFICAS (OPENGL O DIRECTX)

```
float saxpy (
    float2 coords : TEXCOORD0,
    uniform sampler2D textureY,
    uniform sampler2D textureX,
    uniform float alpha ) : COLOR
{
    float result;
    float yval=y_old[i];
    float y = tex2D(textureY,coords);
    float xval=x[i];
    float x = tex2D(textureX,coords);
    y_new[i]=yval+alpha*xval;
    result = y + alpha * x;
    return result;
}
```

$$Y = Y + \text{alpha} * X$$

# PRIMEROS PASOS EN COMPUTACIÓN SOBRE GPUS

## LIMITACIONES QUE IMPIDIERON EL PROGRESO

CURVA DE APRENDIZAJE DE OPENGL/DIRECTX Y ESFUERZO DE TRADUCCIÓN

NECESIDAD DE APRENDER LENGUAJES DE SHADING (CG, GLSL)

SOPORTE DE FLOAT O DOUBLE NO GARANTIZADO

LIMITACIONES EN LOS PATRONES DE ESCRITURA Y LECTURA DE MEMORIA

CARENCIA DE HERRAMIENTAS DE DEPURACIÓN Y CONTROL DE ERRORES

RECURSOS LIMITADOS: MEMORIA, VELOCIDAD, FLEXIBILIDAD...

# ARQUITECTURA CUDA

2007

**CUDA (Compute Unified Device Architecture)** es una plataforma de computación paralela en GPUs desarrollada por nvidia



NVIDIA GEFORCE 8800 GTX (2007)

# ARQUITECTURA CUDA

## ECOSISTEMA

ARQUITECTURA HARDWARE PROPIA

DRIVER ESPECIALIZADO PARA LA GPU

LENGUAJE DE PROGRAMACIÓN FLEXIBLE (BASADO EN C++ INICIALMENTE)

COMPILADOR Y ENTORNO DE DESARROLLO Y DEPURACIÓN

DOCUMENTACIÓN, TUTORIALES, DONACIONES

# INTRODUCCIÓN A CUDA



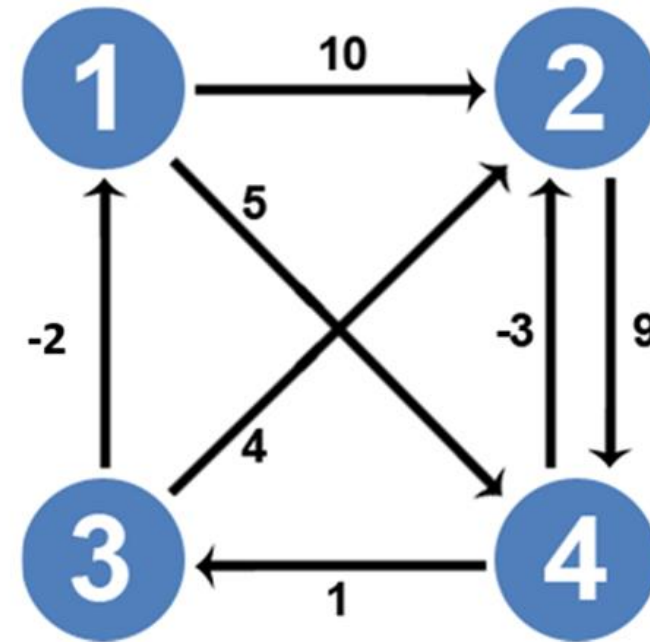
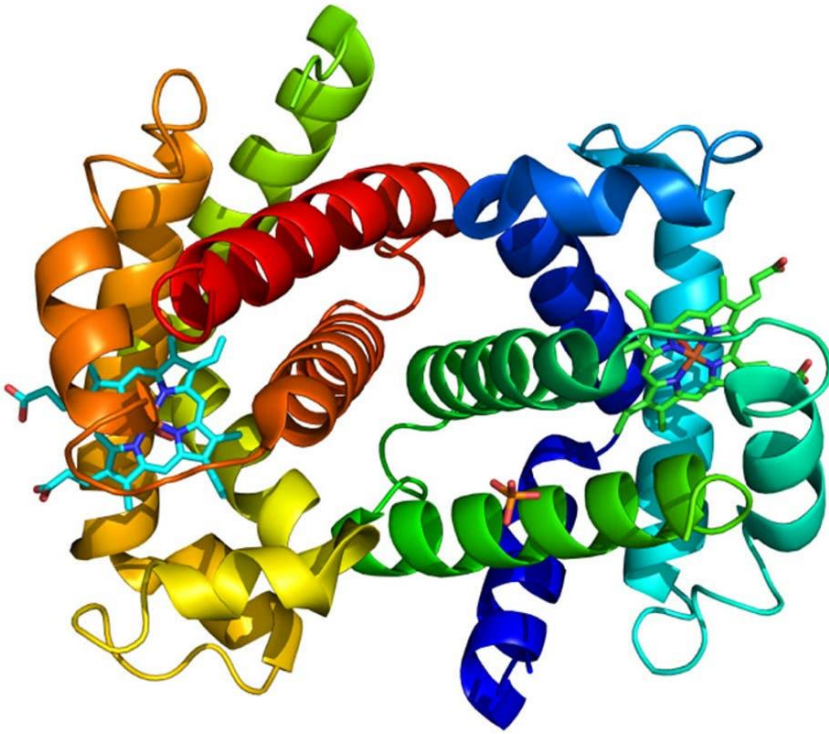
# ¿QUÉ ES CUDA?

NVIDIA RTX 4080 (2022)



# ¿QUÉ ES CUDA?

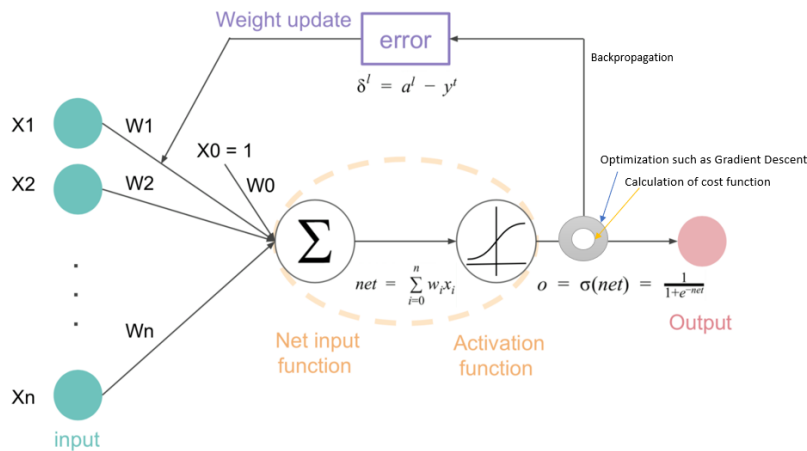
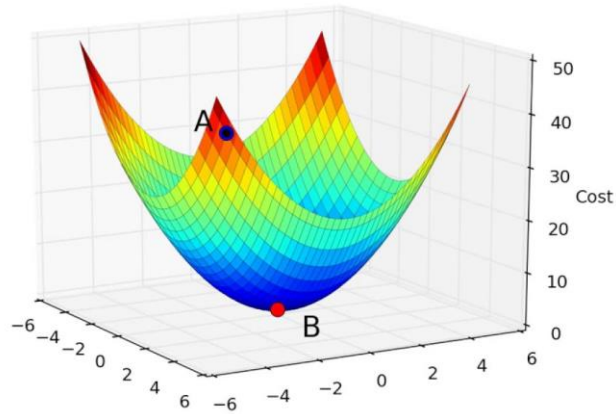
PROTEIN FOLDING, FLOYD WARSHALL ALGORITHM





# ¿QUÉ ES CUDA?

## GRADIENT DESCENT Y BACKPROPAGATION



Machine Learning



Chat GPT  
DALL-E 2  
Whisper



AlphaGo  
AlphaFold 2  
AlphaTensor

# ¿QUÉ ES CUDA?

DEJEMOS QUE LOS CAZADORES DE MITOS LO EXPLIQUEN



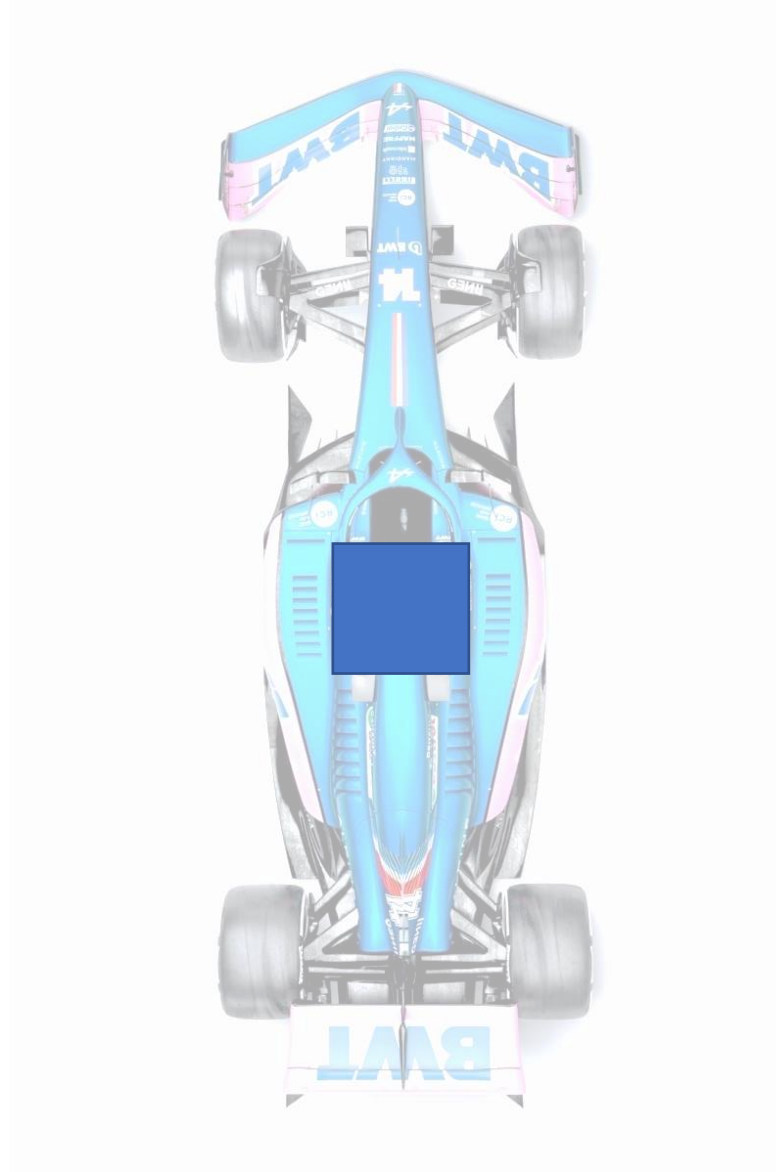
# ¿QUÉ ES CUDA?

## ANALOGÍA



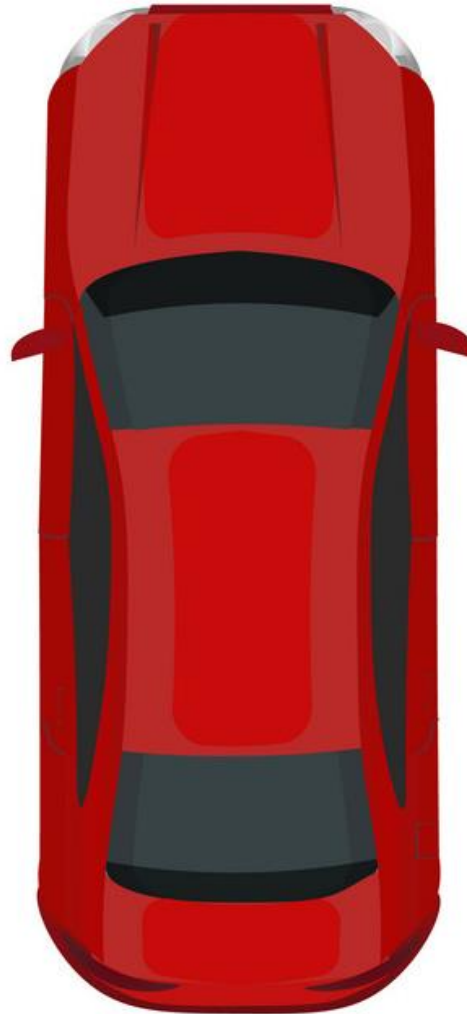
# ¿QUÉ ES CUDA?

## ANALOGÍA



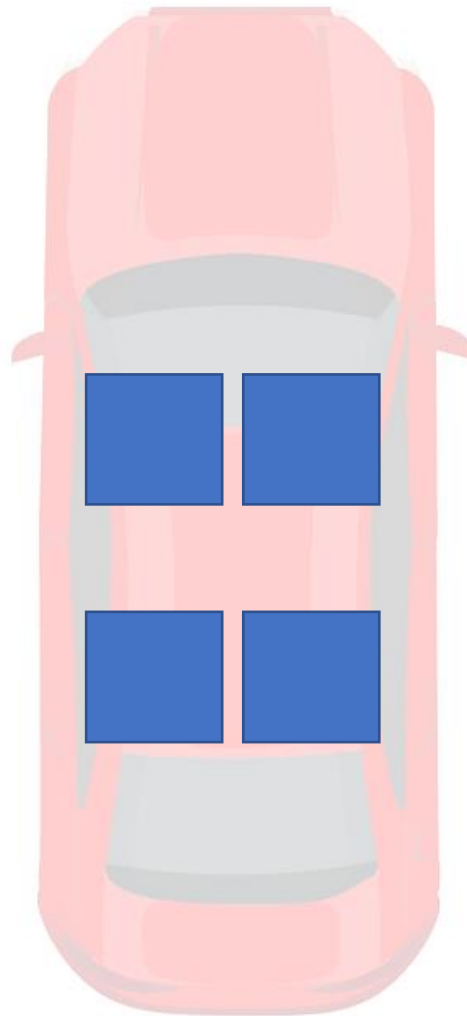
# ¿QUÉ ES CUDA?

## ANALOGÍA



# ¿QUÉ ES CUDA?

## ANALOGÍA



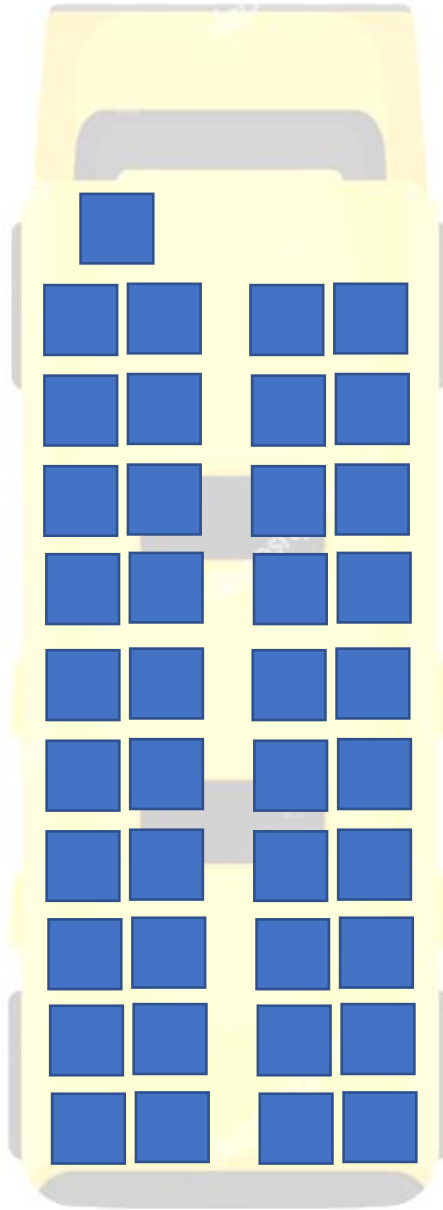
# ¿QUÉ ES CUDA?

## ANALOGÍA



# ¿QUÉ ES CUDA?

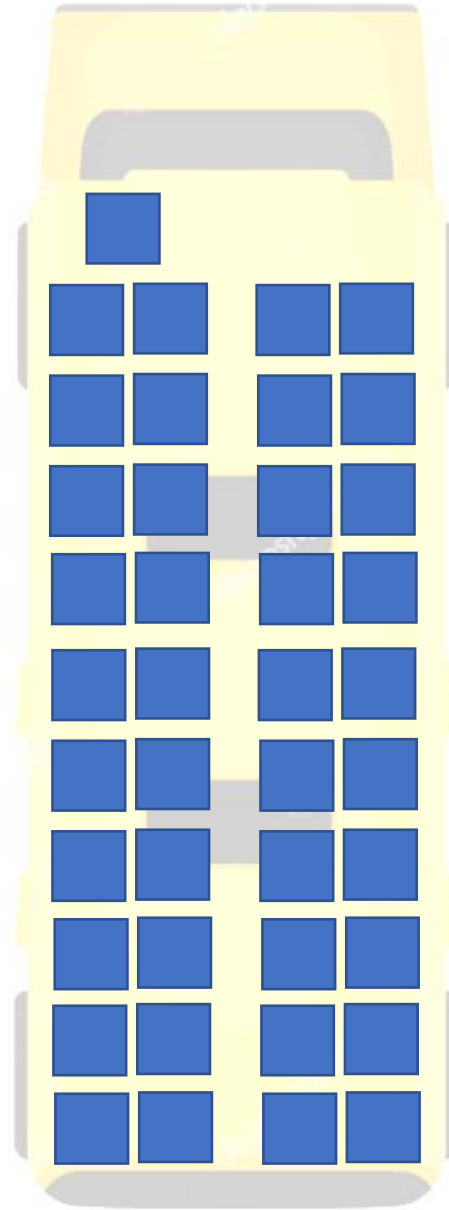
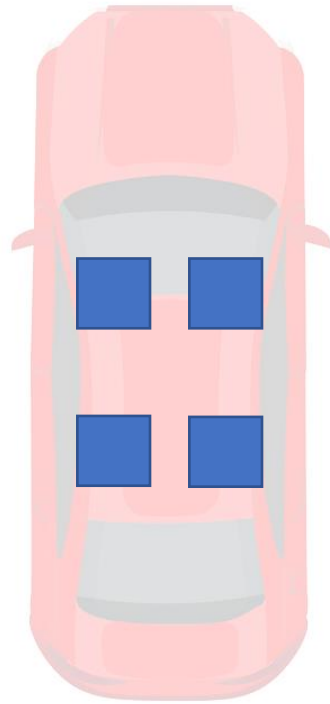
## ANALOGÍA





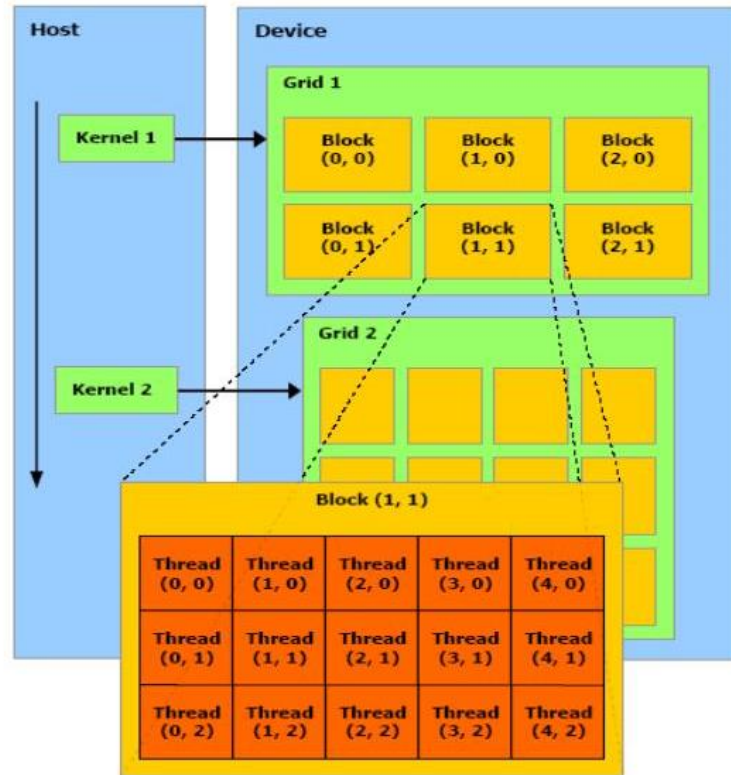
# ¿QUÉ ES CUDA?

## ANALOGÍA



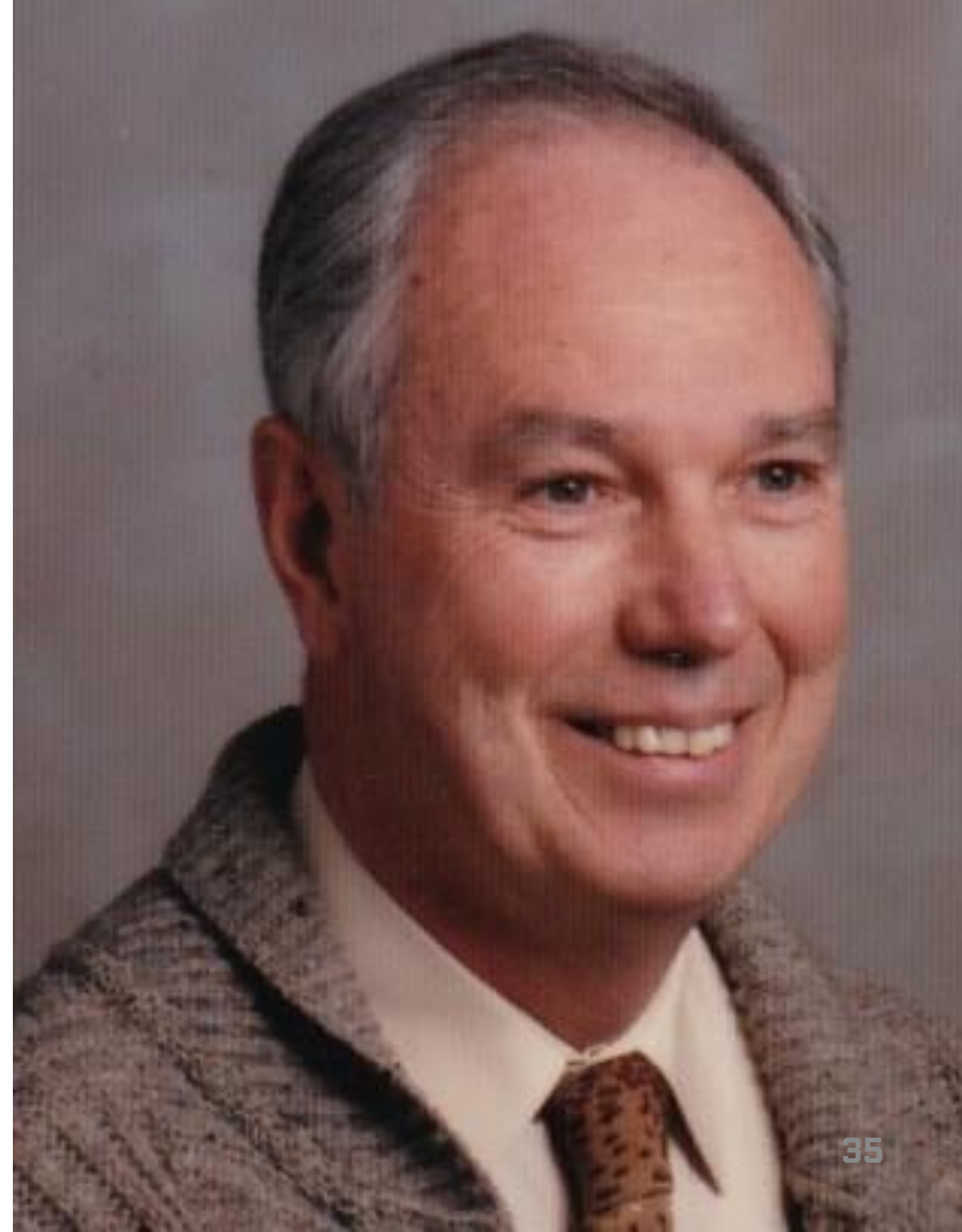
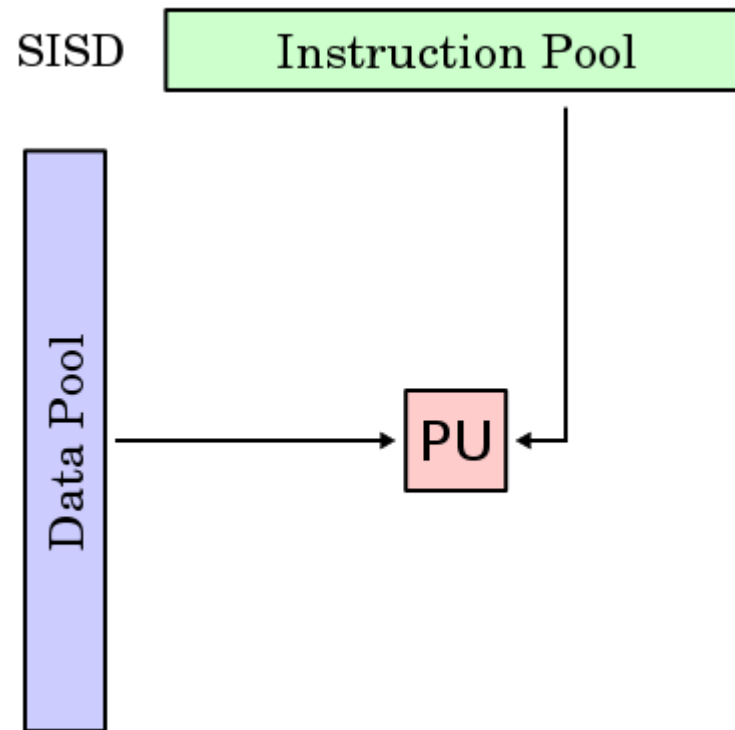
# ARQUITECTURA HARDWARE CUDA

SUBTÍTULO QUE NADIE LEE



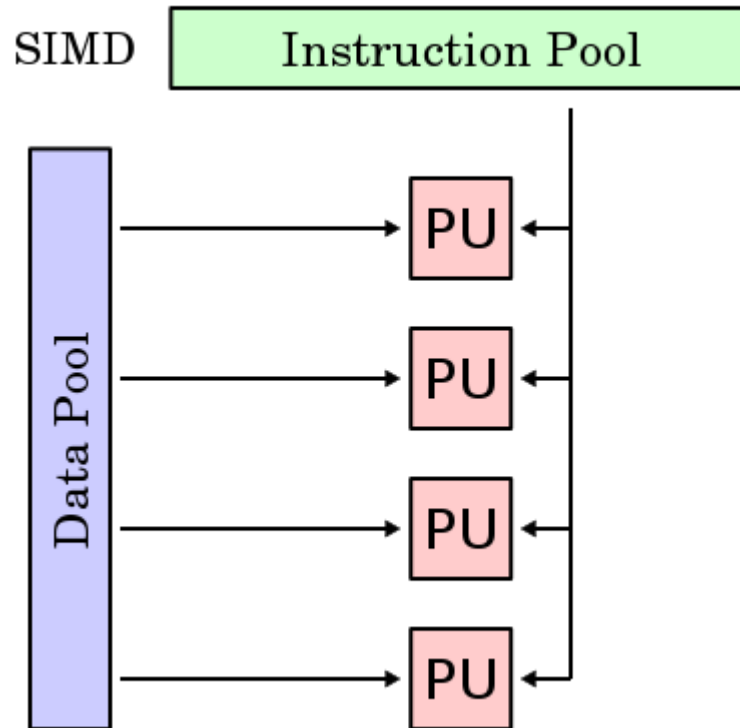
# ARQUITECTURA HARDWARE

## TAXONOMÍA DE FLYNN (SISD)



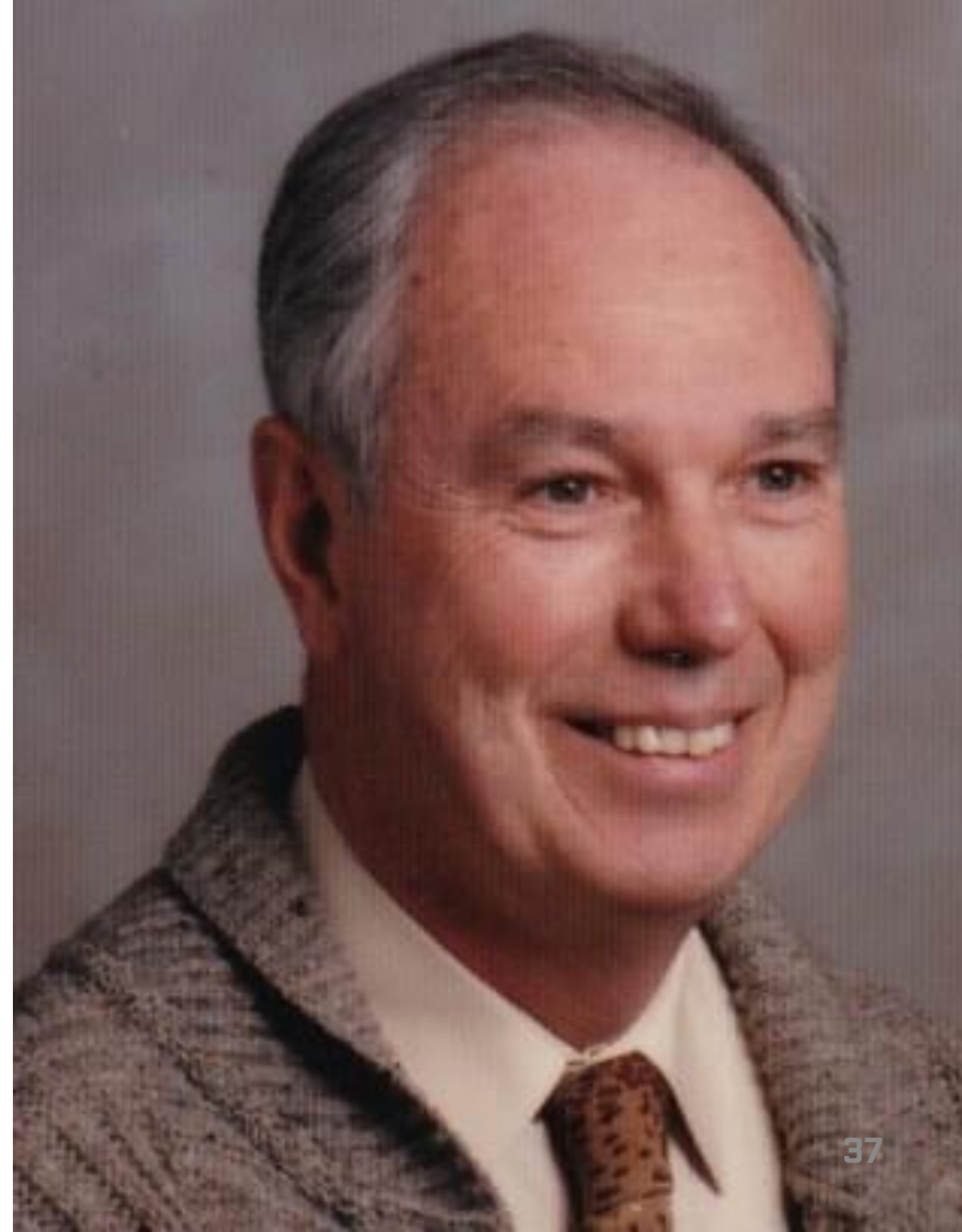
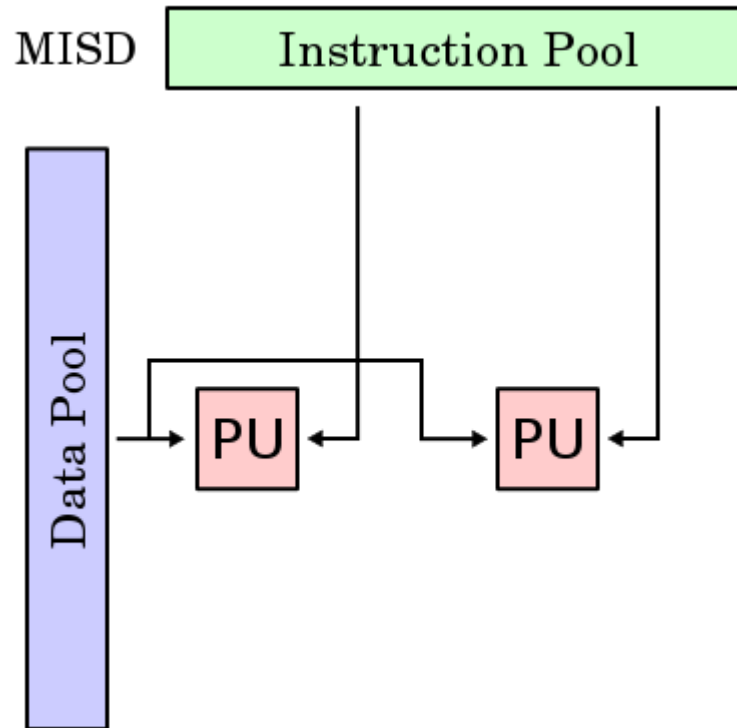
# ARQUITECTURA HARDWARE

## TAXONOMÍA DE FLYNN (SIMD)



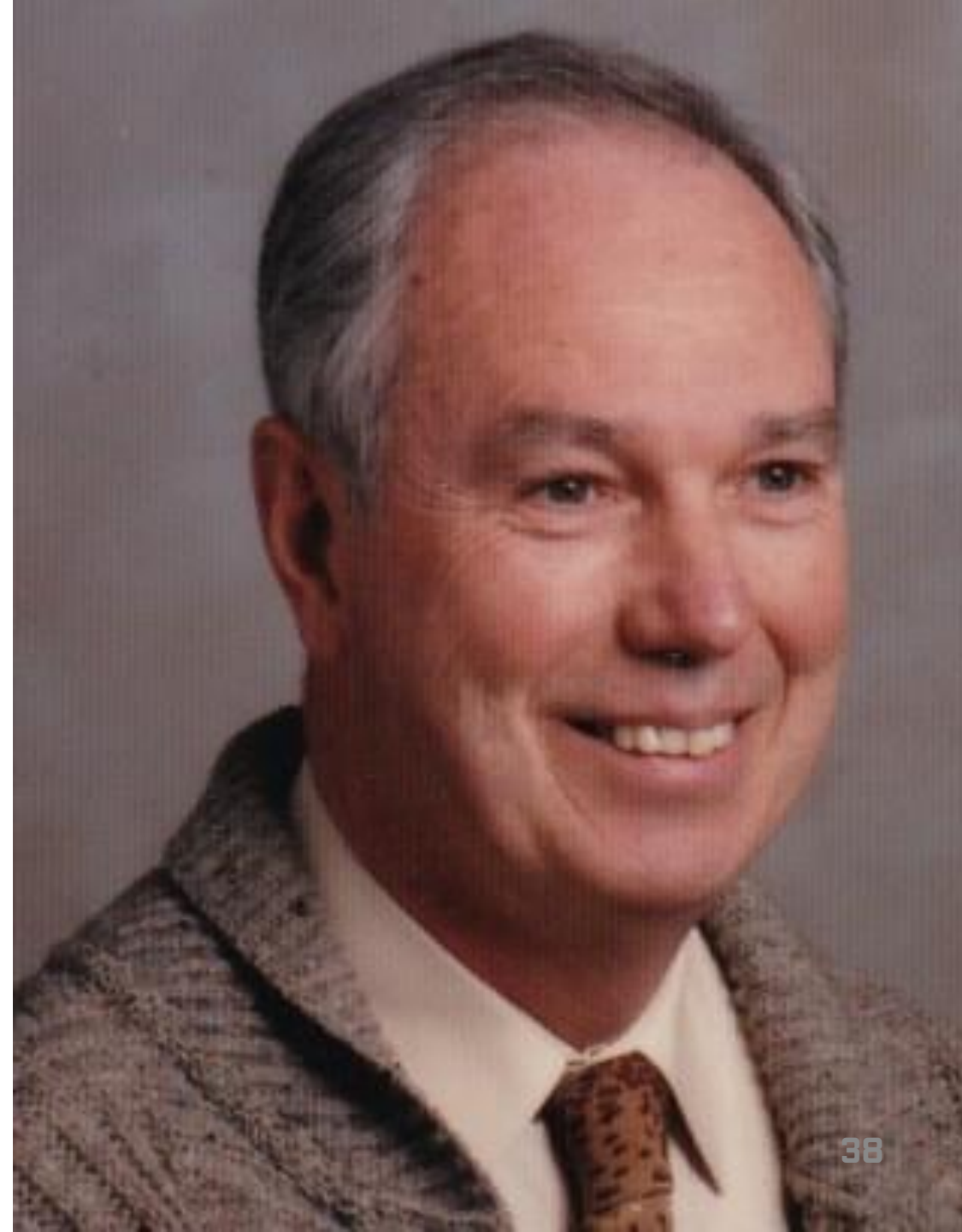
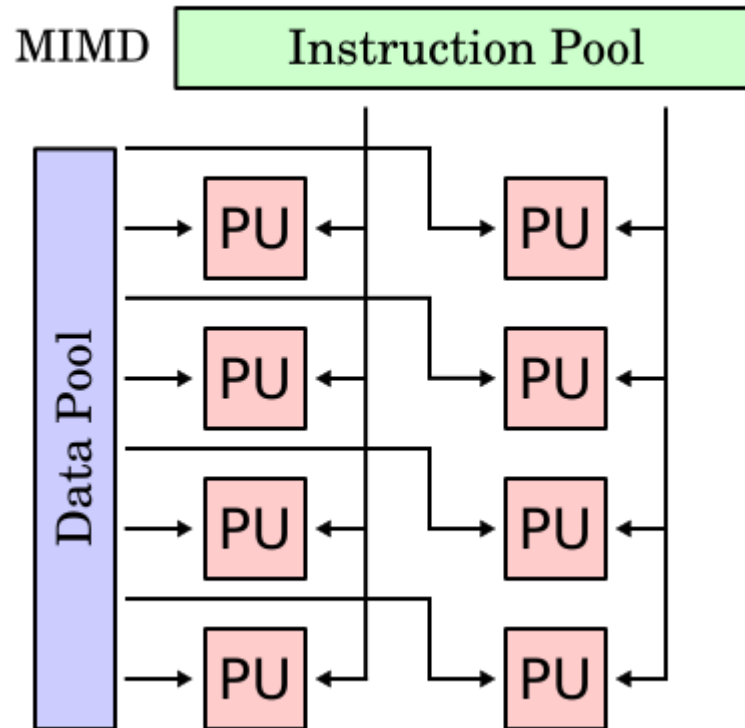
# ARQUITECTURA HARDWARE

## TAXONOMÍA DE FLYNN (MISD)



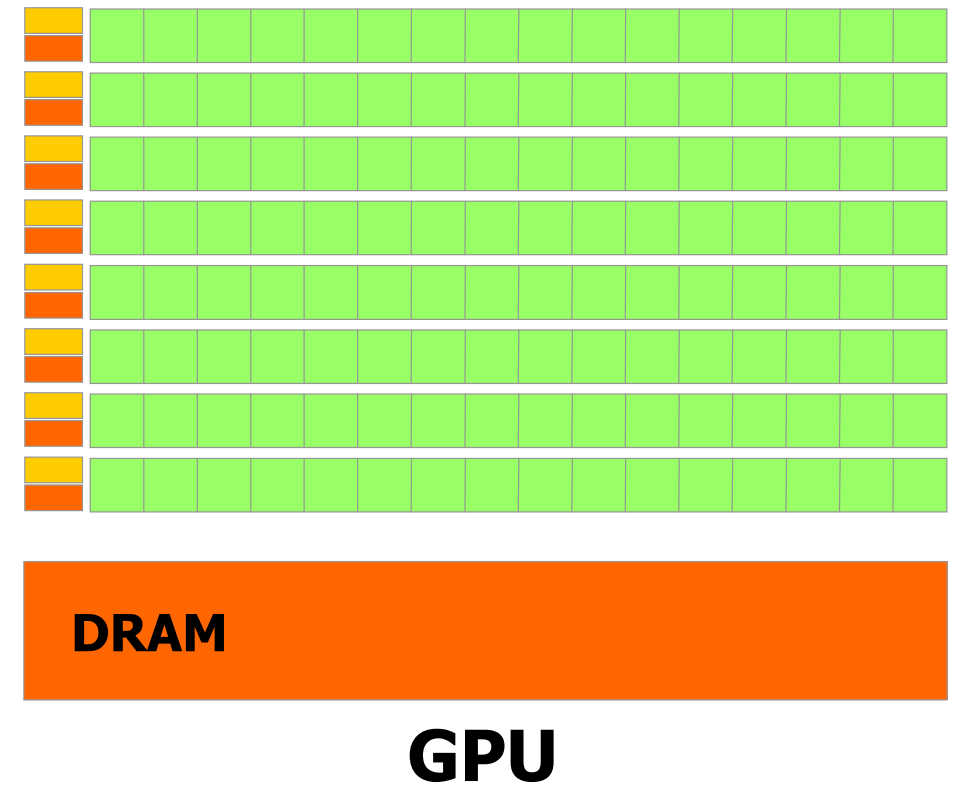
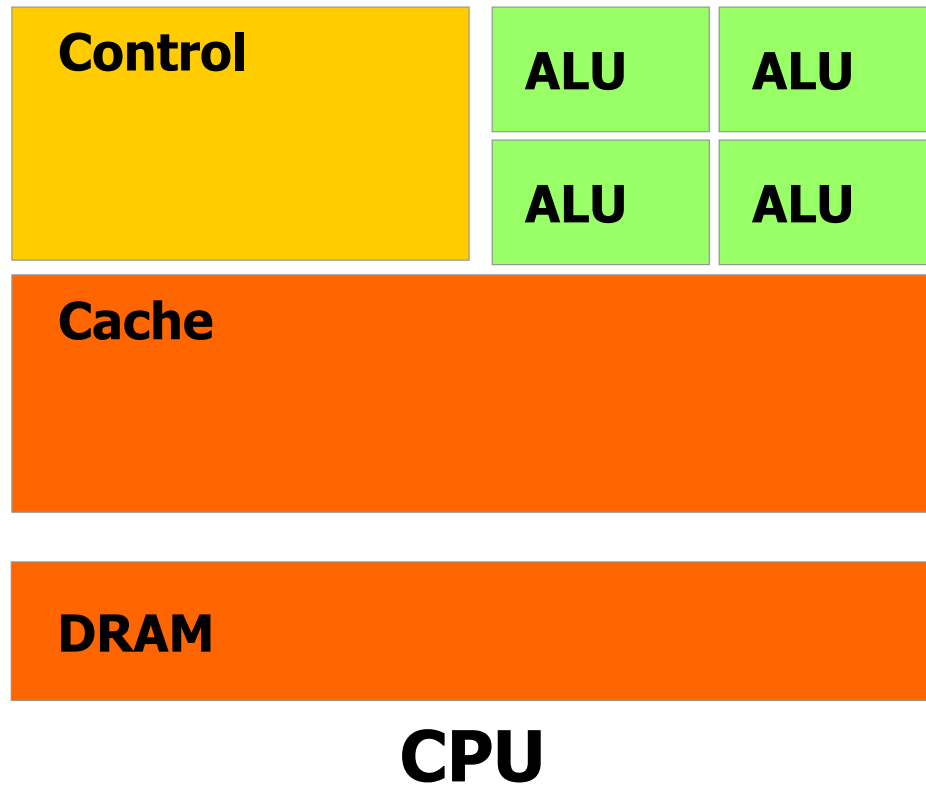
# ARQUITECTURA HARDWARE

## TAXONOMÍA DE FLYNN (MIMD)



# ARQUITECTURA HARDWARE

## DIFERENCIAS ENTRE CPU Y GPU



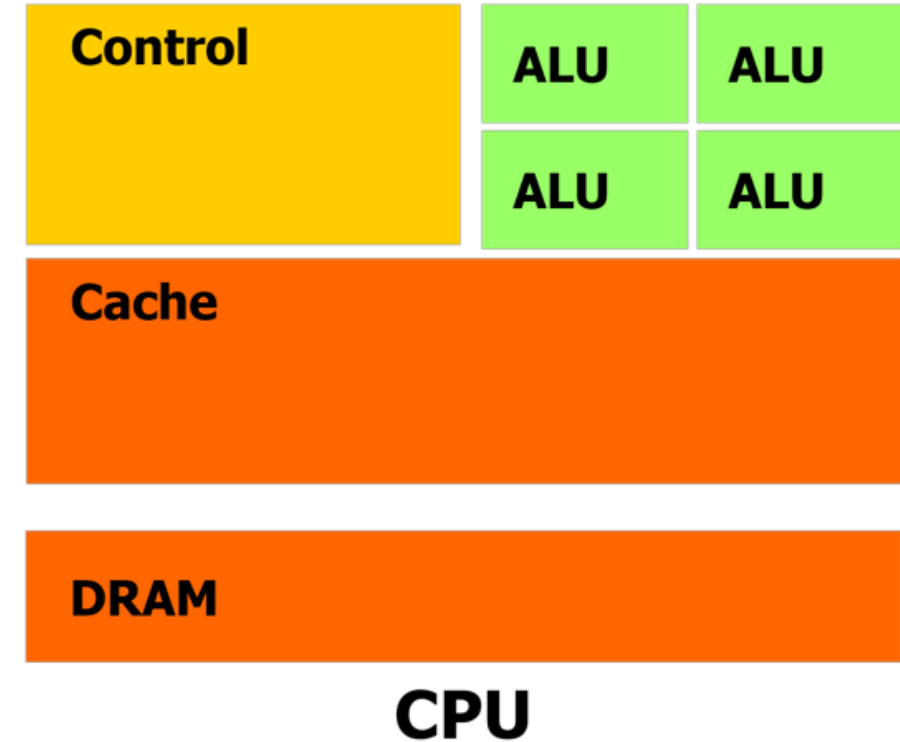


# ARQUITECTURA HARDWARE

## DIFERENCIAS ENTRE CPU Y GPU

### CPU

- **Caches grandes**
  - Permiten bajar la latencia en los accesos a memoria.
- **Unidad de control compleja**
  - Branch prediction
  - Data forwarding
- **ALUs complejas**
  - Reducen latencia de las operaciones



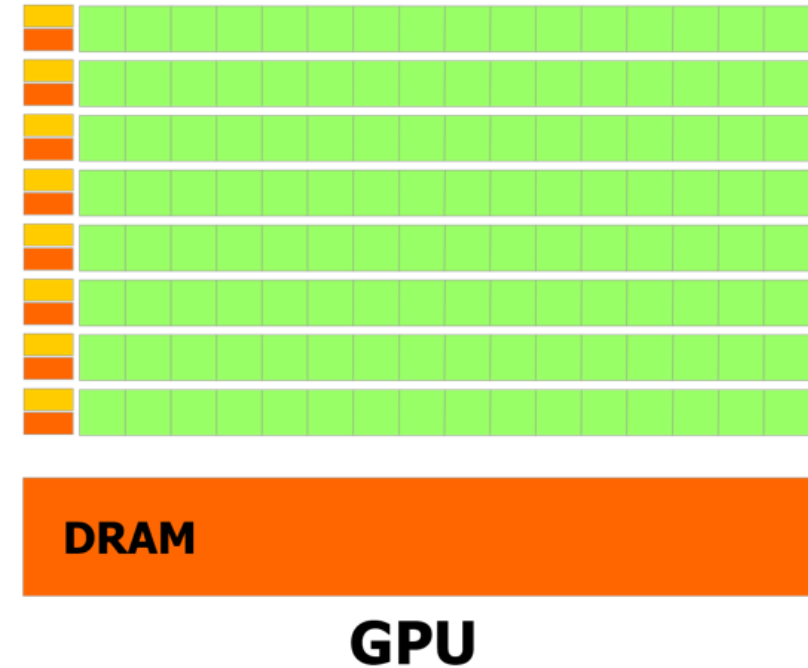


# ARQUITECTURA HARDWARE

## DIFERENCIAS ENTRE CPU Y GPU

### GPU

- **Caches pequeñas**
  - Potenciar ancho banda memoria.
- **Unidad de control simple**
  - No Branch prediction
  - No Data forwarding
- **Muchas ALUs simples**
  - Altamente segmentadas para aumentar el ancho de banda de la memoria
- **Requiere un número muy elevado de hilos para ocultar las latencias**



# ARQUITECTURA HARDWARE

## DIFERENCIAS ENTRE CPU Y GPU

- **Los hilos en la GPU son muy ligeros.**
  - Se necesita muy poco tiempo para crear/destruir hilos.
- **La GPU necesita muchos hilos para ser eficiente.**
  - Una CPU multi-núcleo solo necesita unos pocos hilos.
- **El tiempo de acceso a memoria en la GPU es alto.**
  - En la CPU el ancho de banda es menor.

# ARQUITECTURA HARDWARE

## COMPARACIÓN ENTRE CPU Y GPU

- Intel Core i7-12700

- 12 núcleos MIMD
- Pocos registros, cache multi-nivel
- **70 GB/s** ancho de banda hacia la memoria principal

- NVIDIA GTX4080

- 9728 núcleos, organizados en 76 unidades SM cada una con 128 núcleos.
- Muchos registros, inclusión cachés nivel 1 y 2.
- 5 GB/s ancho de banda hacia el procesador HOST.
- **700 GB/s** ancho de banda memoria tarjeta gráfica.

# ARQUITECTURA HARDWARE

## COMPONENTES BÁSICOS DE UNA GPU CUDA

INTERFAZ CON EL HOST (CPU) QUE CONECTA LA GPU AL BUS PCI EXPRESS

0-2 COPY ENGINES PARA TRANSFERENCIAS DE MEMORIA ASÍNCRONAS

INTERFAZ DE MEMORIA DRAM QUE CONECTA A LA GPU CON SU MEMORIA INTERNA

CIERTO NÚMERO DE GRAPHICS PROCESSING CLUSTERS (GPCS) CON STREAMING MULTIPROCESSORS (SMS) O PROCESADORES DE GENERACIÓN ACTUAL

# ARQUITECTURA HARDWARE

## STREAMING MULTIPROCESSORS

INSTRUCTION CACHE/DECODER

SCHEDULER

CUDA CORES

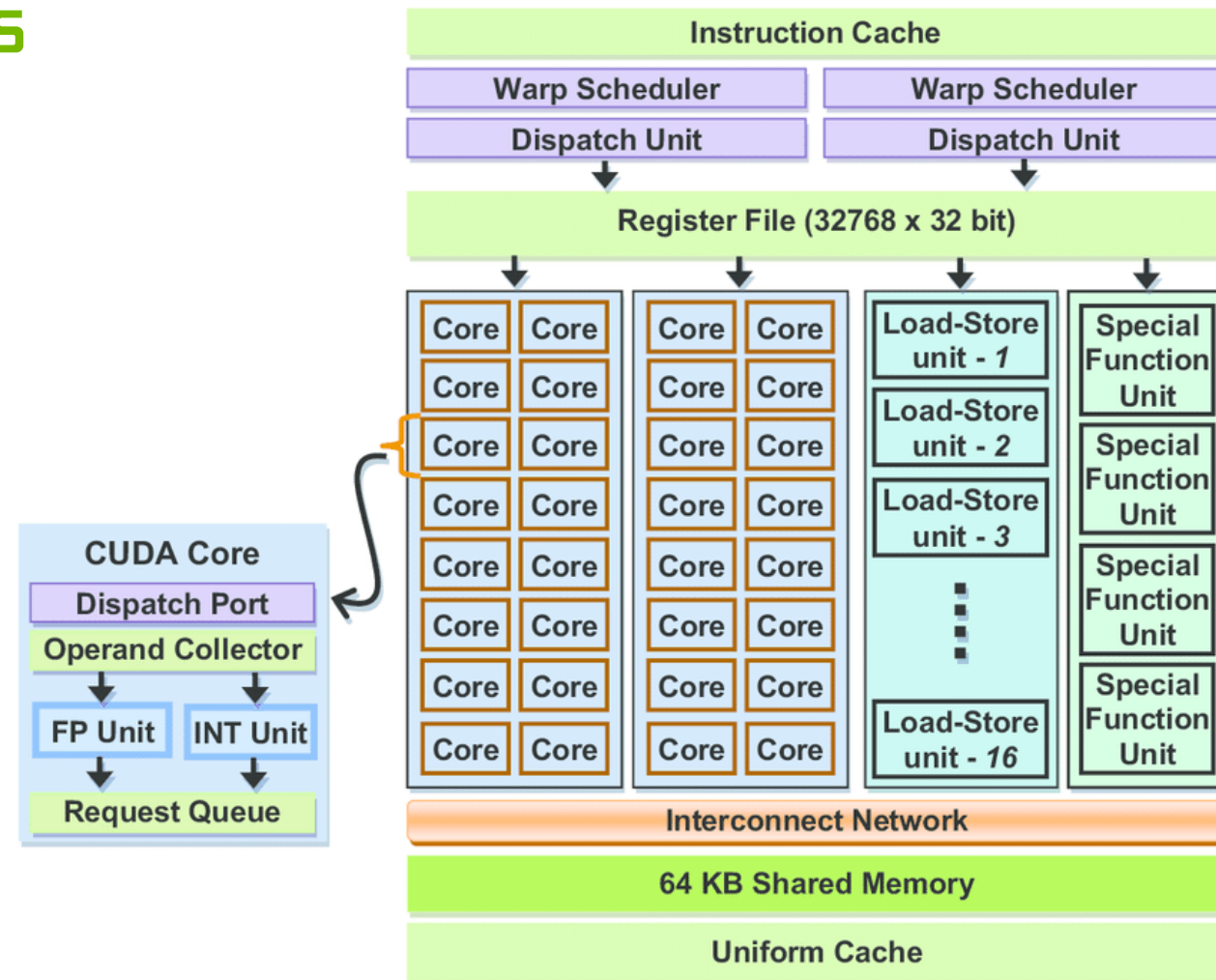
UNIDADES LOAD-STORE

UNIDADES SFU

MEMORIA COMPARTIDA

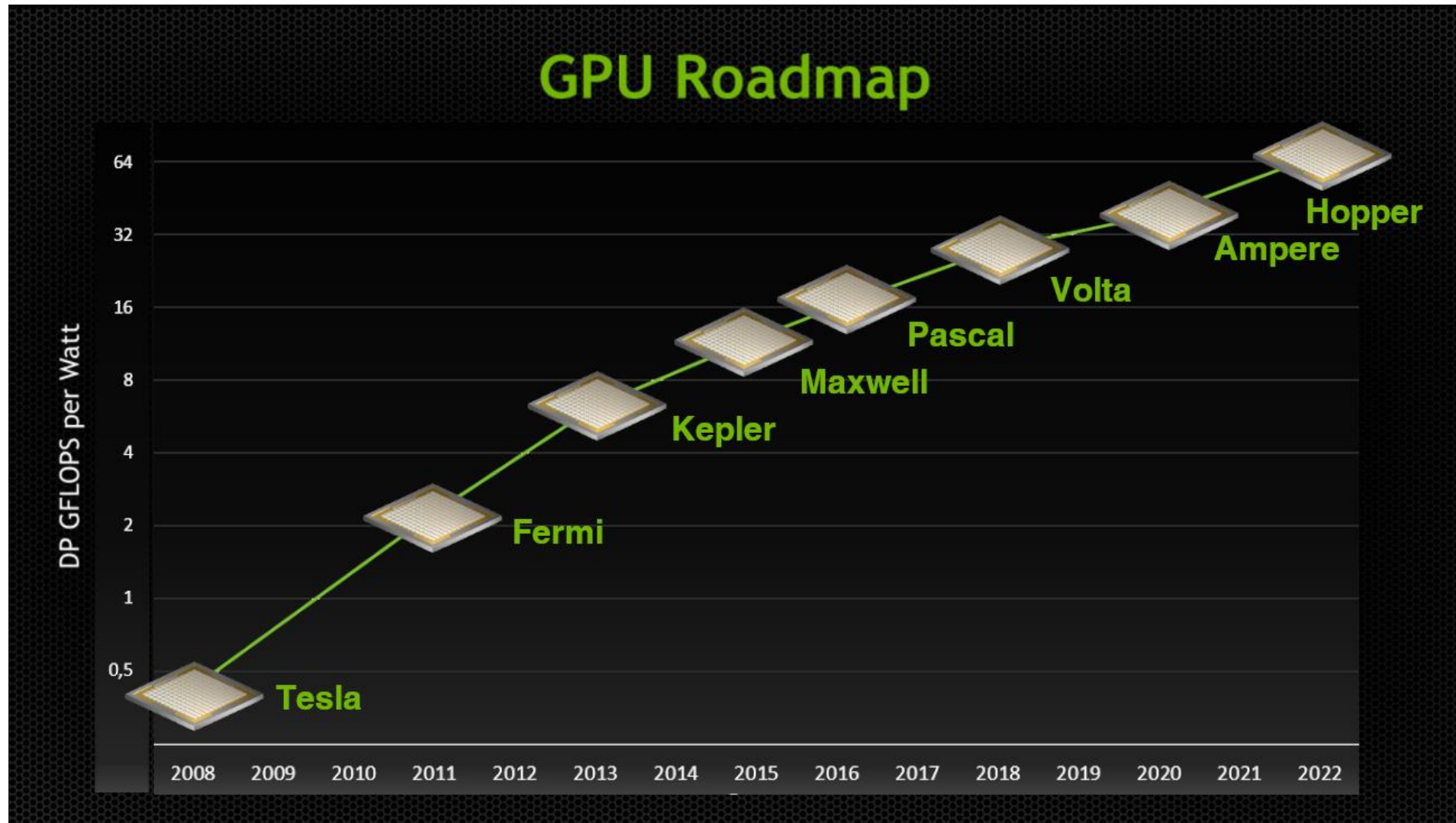
CACHÉS DE TEXTURAS

ARCHIVO DE REGISTROS



# ARQUITECTURA HARDWARE

## EVOLUCIÓN DE MICROARQUITECTURAS

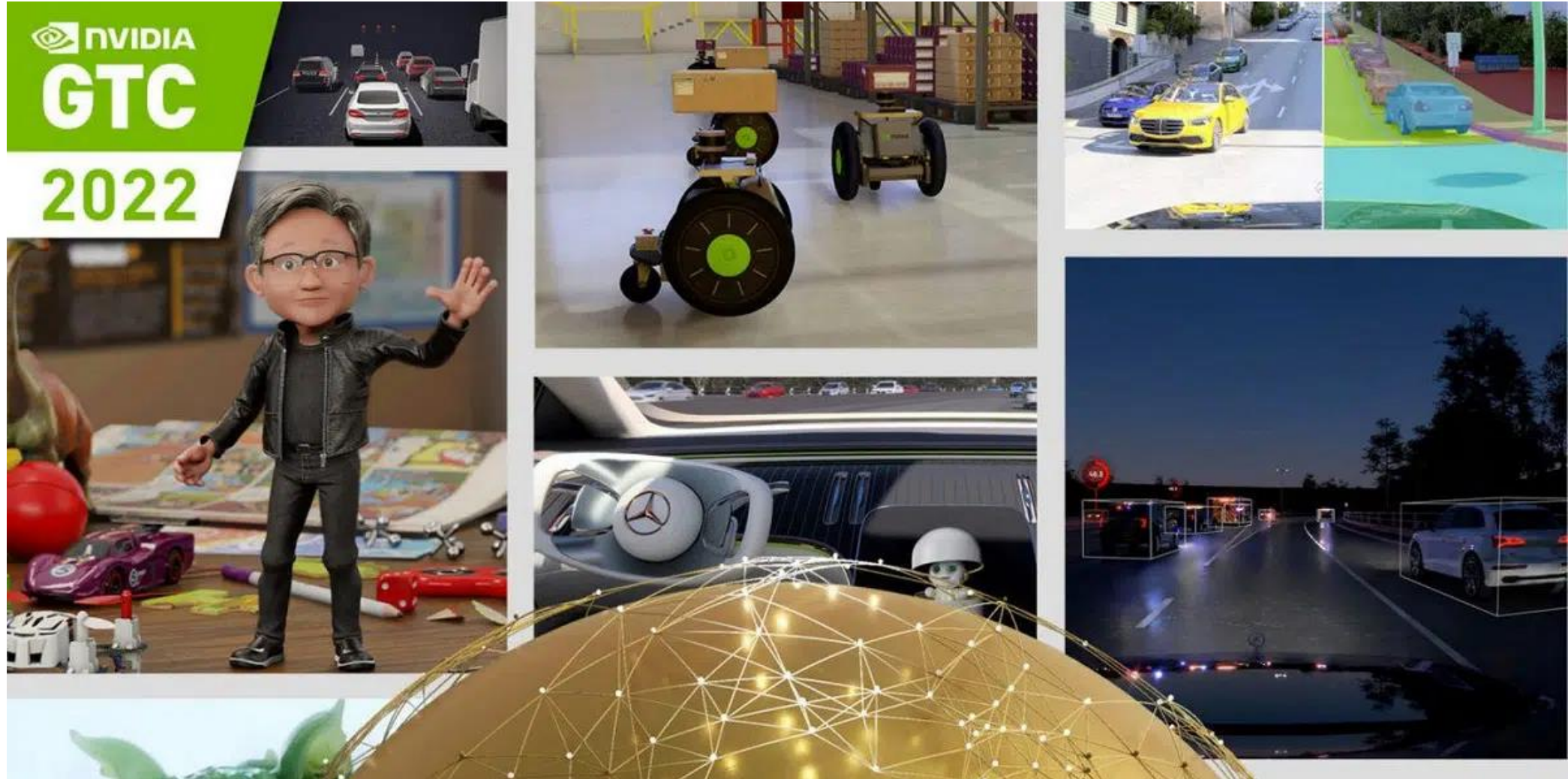




# ARQUITECTURA HARDWARE

GPU TECHNOLOGY CONFERENCE 2022

March 20-23, 2023



# ARQUITECTURA SOFTWARE

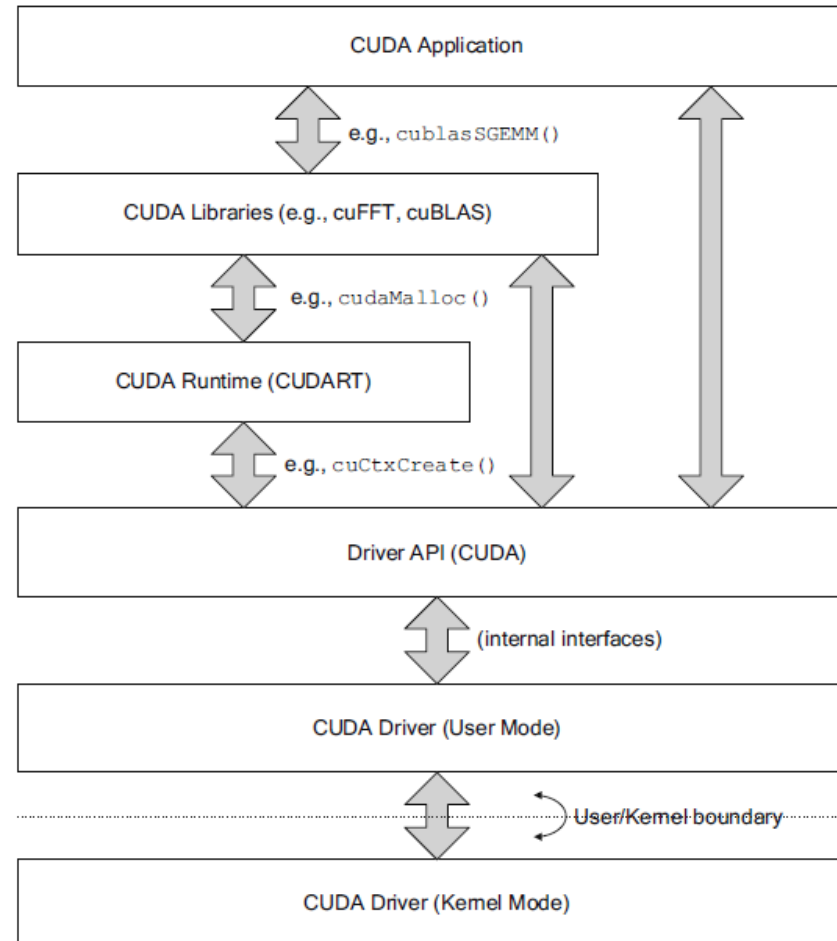
## CUDA

- **CUDA (Compute Unified Device Architecture)**
  - Hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por NVIDIA.
- **Basado en C/C++** con algunas extensiones
- Soporte C++ , Fortran. Wrappers para otros lenguajes: .NET, Python, Java...
- **Gran cantidad de ejemplos** y buena documentación, lo cual reduce la curva de aprendizaje para aquellos con experiencia en lenguajes como OpenMPI y MPI.
- Extensa comunidad de usuarios en los **foros de NVIDIA**.



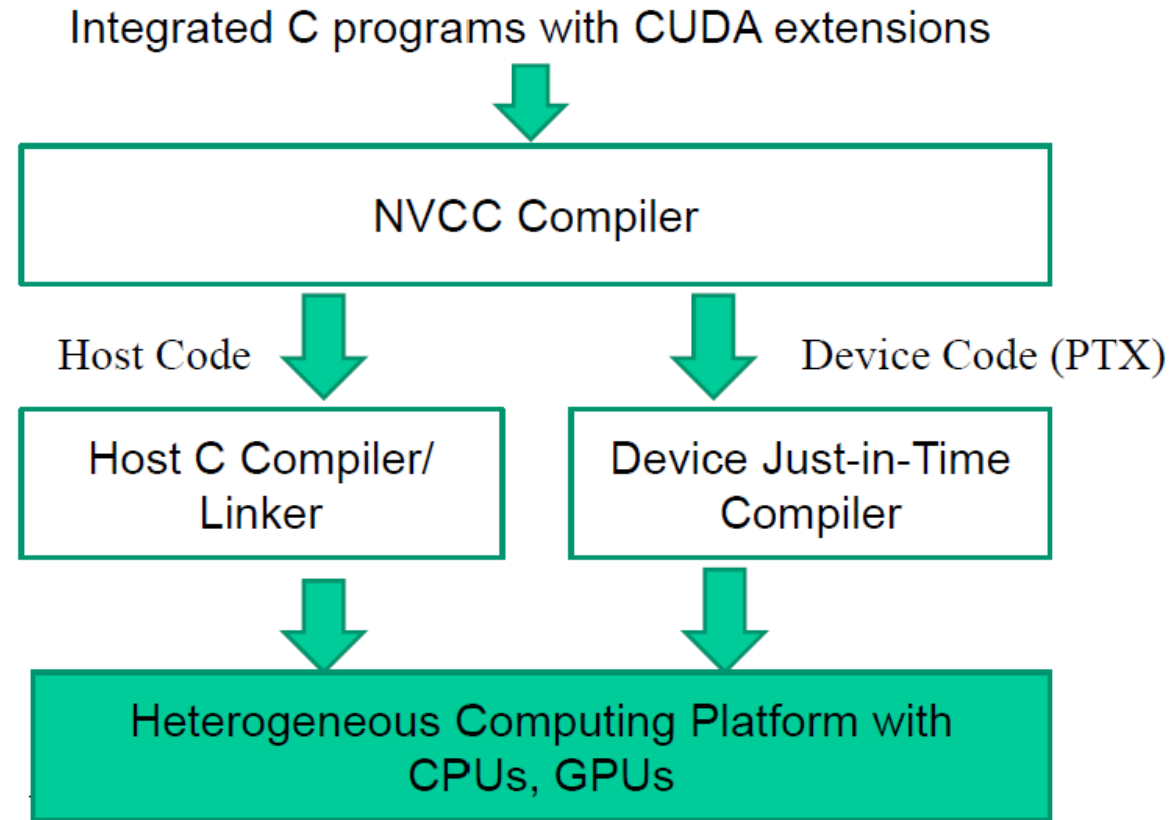
# ARQUITECTURA SOFTWARE

## STACK CUDA



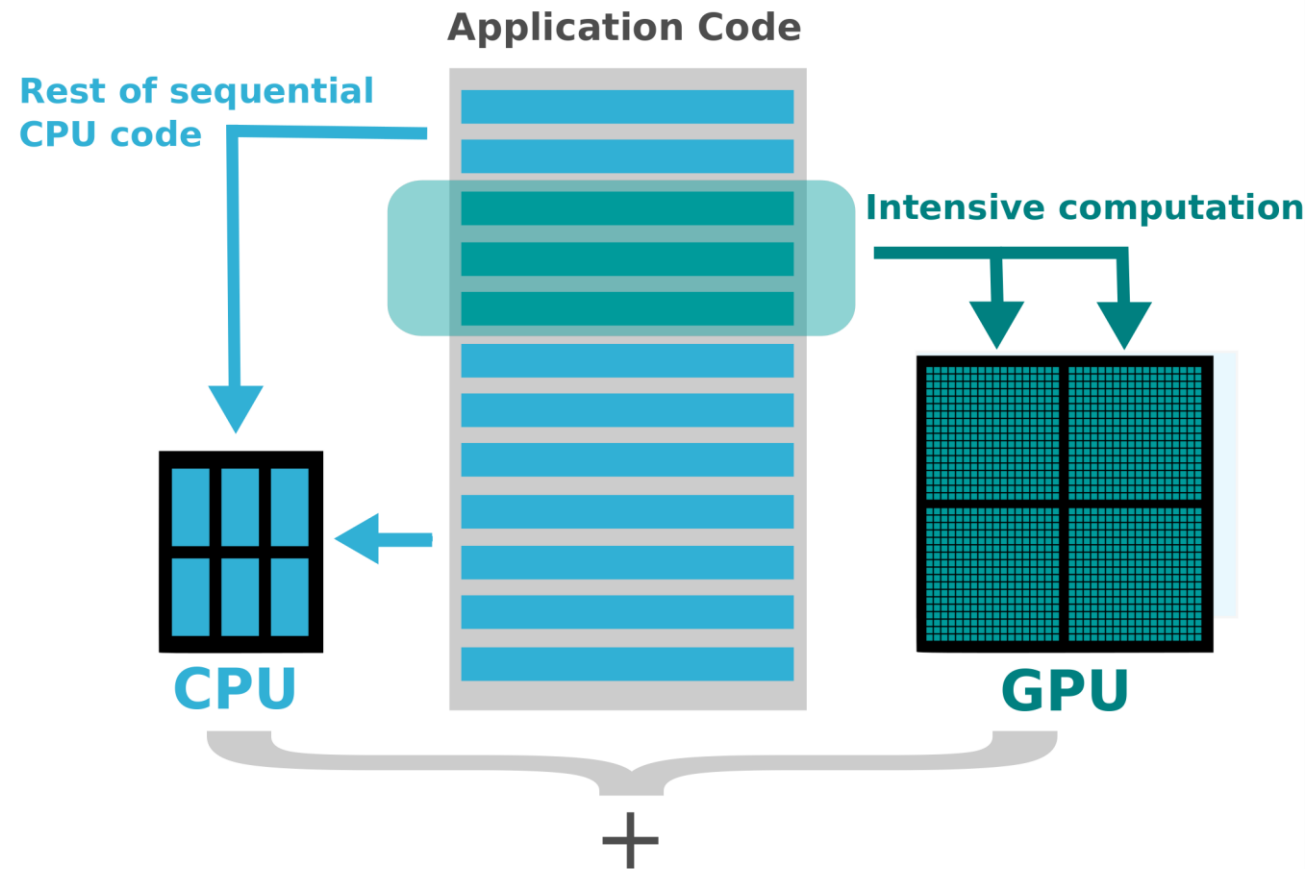
# ARQUITECTURA SOFTWARE

## COMPILACIÓN



# ARQUITECTURA SOFTWARE

## COMPUTACIÓN HETEROGÉNEA



# ARQUITECTURA SOFTWARE

## CONCEPTOS BÁSICOS

**HILO:** es la unidad de ejecución en la GPU, que puede realizar cálculos y acceder a la memoria.

**KERNEL:** es una función que al ejecutarse lo hará en una gran cantidad de hilos y en la GPU.

**BLOCK:** es una agrupación de hilos en 1D, 2D o 3D. Cada bloque se ejecuta sobre un único SM, pero un SM puede tener asignados varios bloques para ejecución.

**GRID:** es una forma de estructurar los bloques, bien en 1D, 2D o 3D

# ARQUITECTURA SOFTWARE

## CONCEPTOS BÁSICOS

### INVOCACIÓN DE KERNEL:

`kernel_routine<<<grid_dim, block_dim>>> (args...);`

- **gridDim** : número de bloques. Tamaño del grid.
- **blockDim**: número de hilos que se ejecutan dentro de un bloque.
- **Args**: número limitado de argumentos, normalmente punteros a memoria de la GPU.

### TAMAÑOS DE BLOQUE Y MALLA DEFINIDOS CON DIM3:

```
dim3 block_dim (32, 32, 1); // 1024 hilos en bloque de 32 x 32 x 1 (2D)
dim3 grid_dim (4, 4, 1); // 16 bloques en malla de 4 x 4 x 1 (2D)
```

# ARQUITECTURA SOFTWARE

## CONCEPTOS BÁSICOS

**CADA HILO EJECUTA UNA COPIA DEL KERNEL, Y DISPONE DE LA SIGUIENTE INFORMACIÓN:**

### **VARIABLES PASADAS COMO ARGUMENTO**

- PUNTEROS A MEMORIA GPU
- VARIABLES SIMPLES POR VALOR (INT, FLOAT, CHAR, LÍMITE DE TAMAÑO)

### **CONSTANTES GLOBALES EN MEMORIA GPU**

### **VARIABLES ESPECIALES (DIM3) PARA IDENTIFICAR AL HILO:**

**gridDim** (tamaño de la malla)

**blockDim** (tamaño de los bloques)

**blockIdx** (identificador de bloque) LOCAL PARA CADA BLOQUE

**threadIdx** (identificador de hilo) LOCAL PARA CADA BLOQUE



# ARQUITECTURA SOFTWARE

## FLUJO DE EJECUCIÓN

En el nivel más alto encontramos un proceso sobre la CPU (Host) que realiza los siguientes pasos:

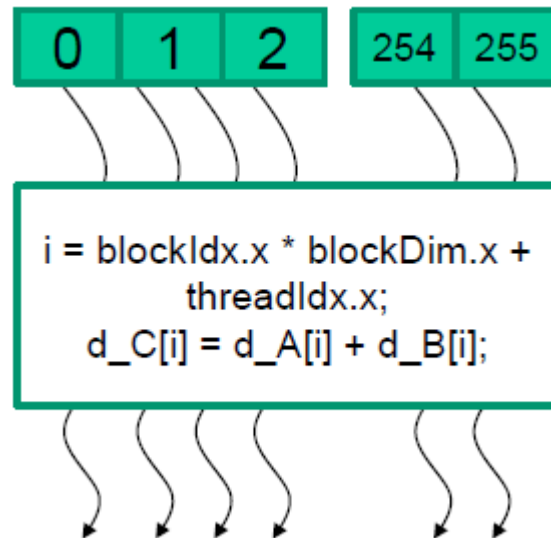
1. Inicializa GPU
2. Reserva memoria en la parte host y device
3. Copia datos desde el **host** hacia la memoria **device**
4. Lanza la ejecución de múltiples copias del **kernel**
5. Copia datos desde la memoria **device** al **host**
6. Se repiten los pasos 3-5 tantas veces como sea necesario
7. Libera memoria y finaliza la ejecución proceso maestro

# ARQUITECTURA SOFTWARE

## EJECUCIÓN DE UN KERNEL

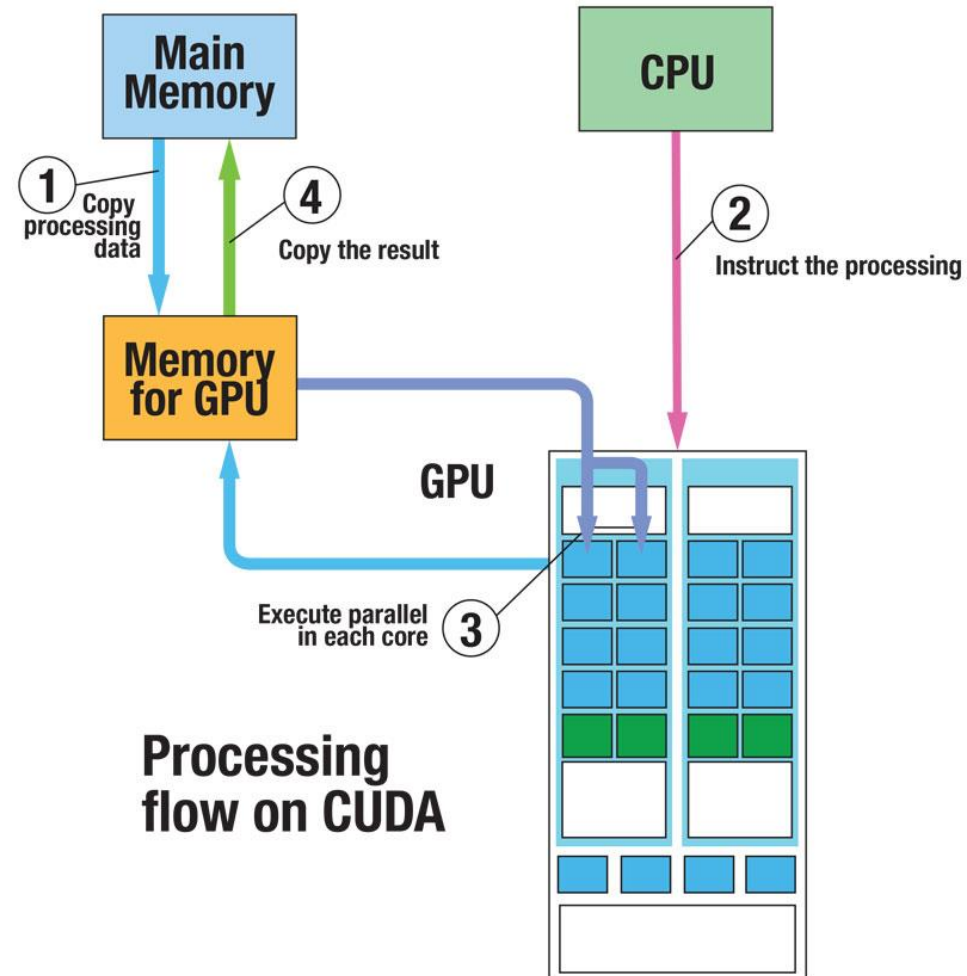
### Un kernel CUDA se ejecuta sobre un grid de hilos:

- Todos los threads del grid ejecutan el mismo código
- Cada hilo tiene sus índices y lo utiliza para direccionar la memoria y realizar su lógica.



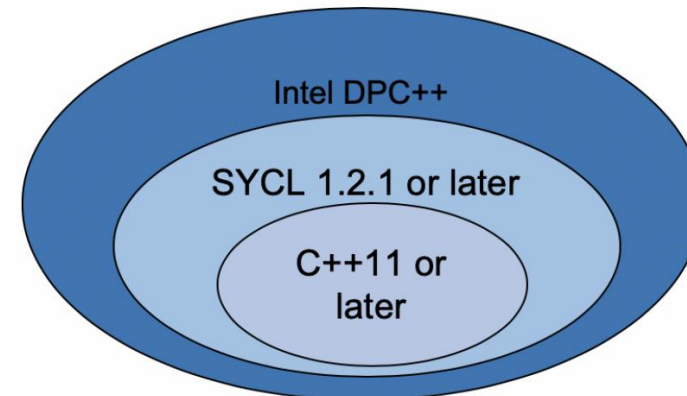
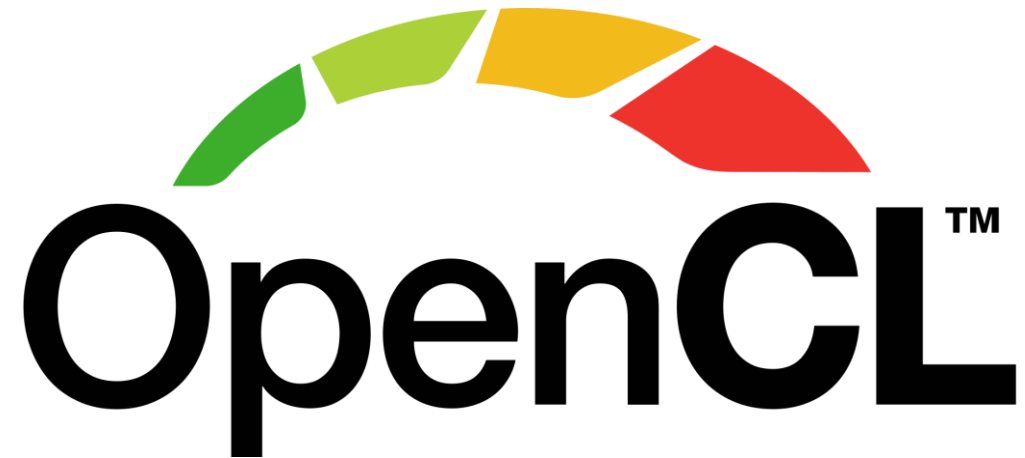
# ARQUITECTURA SOFTWARE

## FLUJO DE EJECUCIÓN



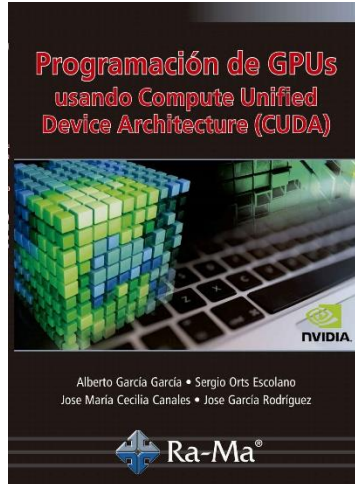
# ALTERNATIVAS

## OTROS MODELOS DE COMPUTACIÓN HETEROGÉNEA



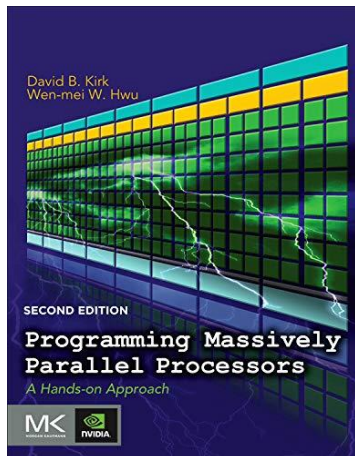
# BIBLIOGRAFÍA RECOMENDADA

## PARA APRENDER UN POQUITO MÁS



### Programación de GPUs Usando Compute Unified Device Architecture (CUDA)

- Alberto García García, Sergio Orts Escolano, José Celilia Canales, José García Rodríguez

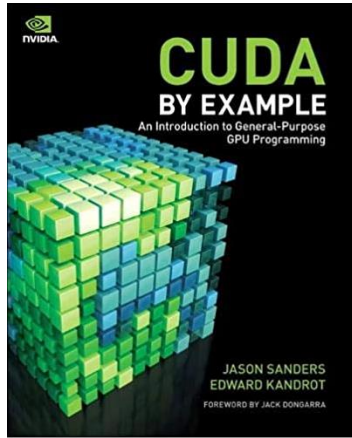


### Programming Massively Parallel Processors (A Hands-on Approach)

- David B. Kirk, Wen-mei W. Hwu

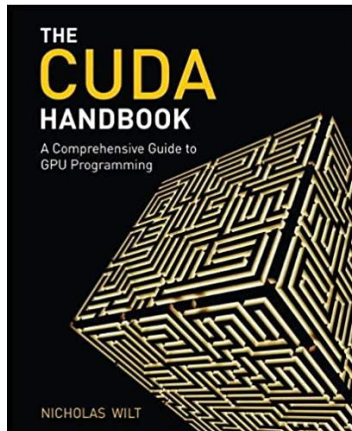
# BIBLIOGRAFÍA RECOMENDADA

## PARA APRENDER UN POQUITO MÁS



### **CUDA BY EXAMPLE (An Introduction to General-Purpose GPU Programming)**

- Jason Sanders, Edward Kandrot



### **THE CUDA HANDBOOK (A Comprehensive Guide to GPU Programming)**

- Nicholas Wilt



# ALGUNA PREGUNTA?

**JORNADA DE DIVULGACIÓN DE APLICACIONES CIENTÍFICAS SOBRE  
PROCESADORES GRÁFICOS Y QUANTUM COMPUTING**

**JGPUQC 2023 - UNIVERSIDAD DE ALICANTE**

Manuel Benavent-Lledó <[mbenavent@dtic.ua.es](mailto:mbenavent@dtic.ua.es)>

David Mulero-Pérez <[dmulero@dtic.ua.es](mailto:dmulero@dtic.ua.es)>

José García-Rodríguez <[jgarcia@dtic.ua.es](mailto:jgarcia@dtic.ua.es)>