



Introduzione a GPU e CUDA

Sistemi Digitali, Modulo 2

A.A. 2024/2025

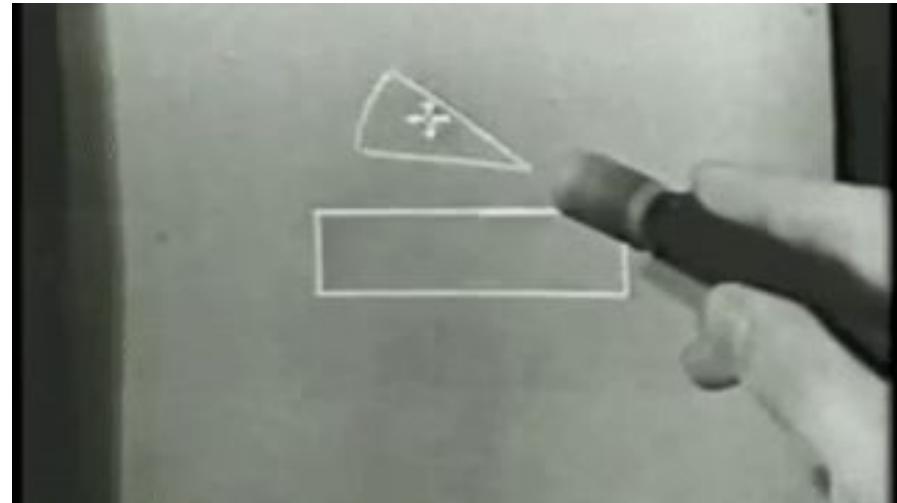
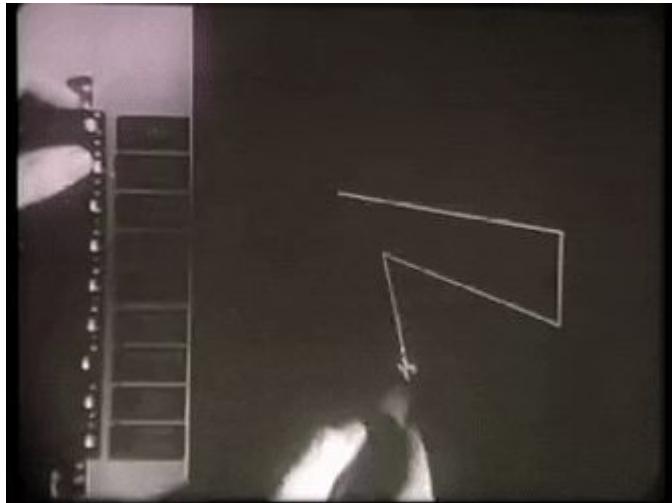
Fabio Tosi, Università di Bologna

Come la Grafica è Diventata Calcolo Parallelo

La Nascita della Computer Grafica

Ivan Sutherland e "Sketchpad" (1963)

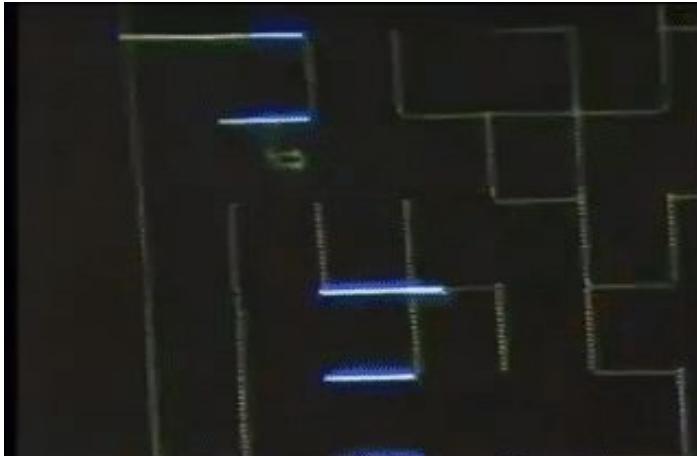
- "Sketchpad", sviluppato da Ivan Sutherland, è considerato il primo programma di **grafica interattiva**, utilizzando un'interfaccia basata su una **penna ottica** per creare immagini su uno schermo.
- Questo progetto pionieristico ha dimostrato le potenzialità della **grafica computerizzata**, aprendo la strada allo sviluppo della computer grafica come campo di studio.



La Nascita della Computer Grafica

La sfida delle risorse computazionali

- Negli anni '60 e '70, la **grafica era gestita direttamente dalla CPU**, che eseguiva sia i calcoli logici sia la generazione delle immagini.
- Questa gestione centralizzata **limitava le capacità di calcolo** della CPU per altri compiti, rallentando l'elaborazione e limitando la complessità delle immagini prodotte.
- La **crescente domanda di grafica** più complessa richiedeva una soluzione più efficiente.



Mouse in The Maze (1961)



Spacewar! (1962)

Primi Passi nell'Accelerazione Grafica

ANTIC di Atari (1977)

- L'ANTIC (Alpha-Numeric Television Interface Controller) fu uno dei primi esempi di **coprocessore grafico**.
- Introdotto da Atari nel 1977 per i suoi **computer a 8-bit**.
- Liberava la CPU dalla gestione della grafica, consentendo **giochi e interfacce più complesse**.
- **Gestiva sprite, scrolling e diverse modalità grafiche**, migliorando l'esperienza visiva.



ANTIC

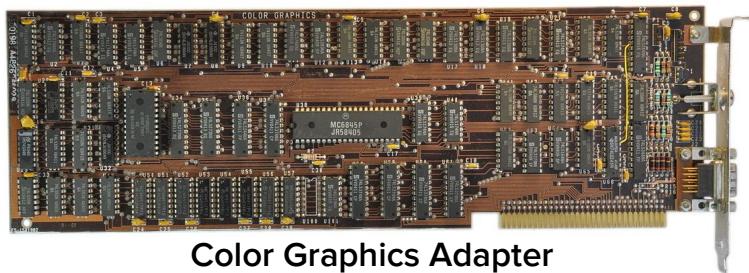


Game Experience

Le Prime Schede Video dedicate e l'Esigenza di Standard

Le Prime Schede Video

- **MDA (Monochrome Display Adapter, 1981)**: Introdotta da IBM, supportava solo testo in modalità monocromatica a una risoluzione di 720x350 pixel.
- **CGA (Color Graphics Adapter, 1981)**: La prima scheda a supportare la grafica a colori, seppur limitata a 4 colori simultanei scelti da una tavolozza di 16. Le risoluzioni disponibili erano 320x200 o 640x200 pixel.
- **EGA (Enhanced Graphics Adapter, 1984)**: Un passo avanti significativo, con una gamma di 16 colori e una risoluzione massima di 640x350 pixel.



Color Graphics Adapter



Il Bisogno di Standard

- La proliferazione di schede video e monitor con specifiche differenti creava enormi problemi di compatibilità.

VGA: Un Predecessore delle GPU

VGA (Video Graphics Array, 1987)

- **Standard VGA:** Introdotto da IBM nel 1987, ha definito le specifiche per schede video e monitor, standardizzando la grafica su PC.
 - *Risoluzione:* 640x480 pixel (modalità standard).
 - *Colori:* Supporto fino a 256 colori simultanei.
 - *Compatibilità:* Retrocompatibile con CGA ed EGA.
- **Controller VGA:** Chip specializzato che implementa lo standard, **gestisce l'output grafico** (ma non i calcoli complessi, gestiti dalla CPU).
- **Scheda Video VGA:** La scheda contenente il controller VGA e la memoria video, destinata a gestire la visualizzazione su monitor.
- **Connettore VGA:** Connettore analogico a 15 pin per la connessione ai monitor.



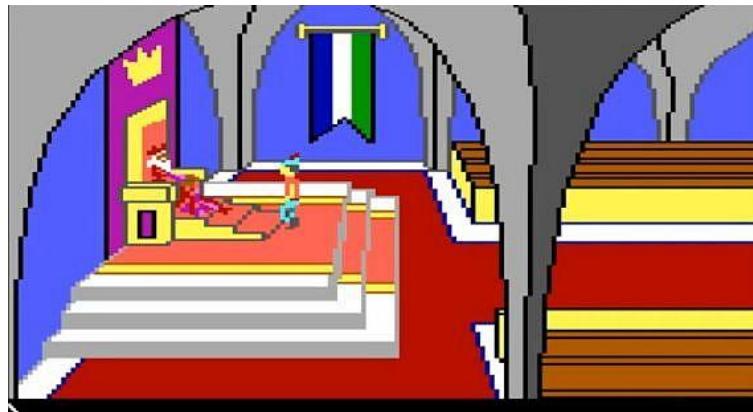
Scheda VGA



Connettore VGA

Limitazioni delle Prime Schede Video

- **Grafica 2D**
 - Le prime schede video, incluse la VGA, erano progettate principalmente per **gestire grafica 2D**, supportando operazioni di base come linee, rettangoli e riempimenti di aree.
- **Basse Risoluzioni**
 - Nonostante la VGA offrisse una risoluzione superiore (fino a 640x480), rimaneva ancora **limitata** per visualizzazioni molto dettagliate.
 - L'esperienza grafica era confinata a **semplici interfacce** e giochi con **grafica di base**.



King's Quest 1 - EGA Mode



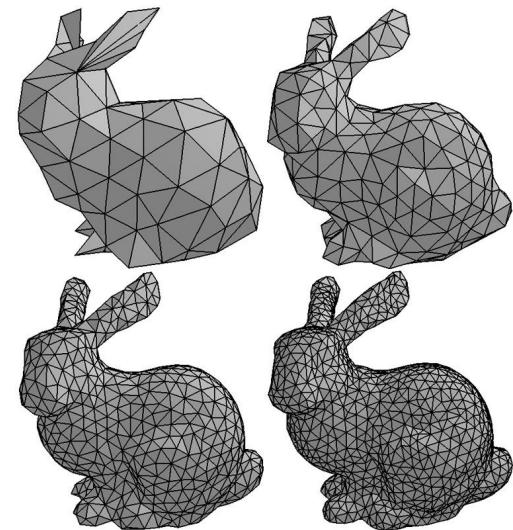
King's Quest 1 - VGA Mode

Funzioni 3D Basilari (metà anni '90)

- Dopo lo standard VGA, la crescente domanda di grafica più realistica nei videogiochi e nelle applicazioni professionali portò allo sviluppo di **funzionalità 3D basilari**.

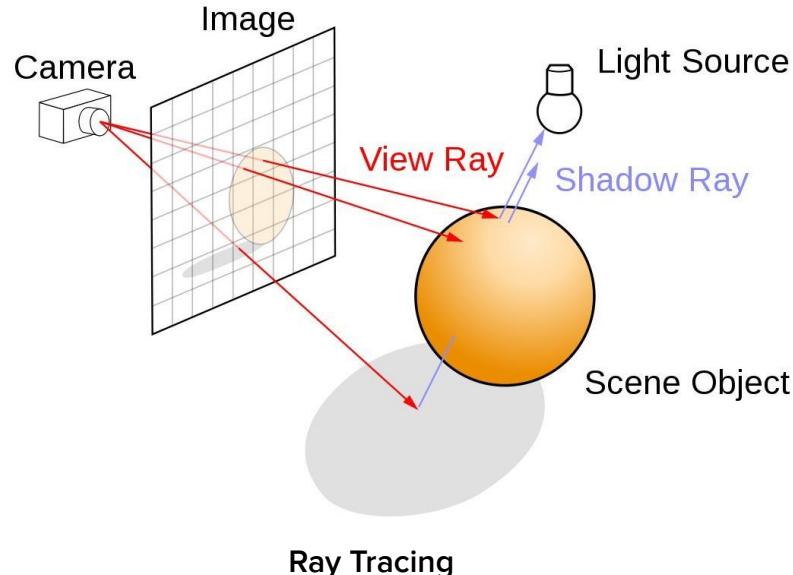
Blocchi Fondamentali della Grafica 3D Moderna

- Triangolazione**
 - Scomposizione di oggetti 3D in triangoli
 - Scopo: Semplificare la rappresentazione di forme complesse
- Rasterizzazione**
 - Conversione di forme vettoriali in pixel
 - Scopo: Renderizzare oggetti 3D su uno schermo 2D
- Texture Mapping**
 - Applicazione di immagini 2D su superfici 3D
 - Scopo: Aggiungere dettagli e realismo.
- Shading**
 - Calcolo dell'illuminazione e del colore delle superfici
 - Scopo: Simulare l'interazione della luce con gli oggetti 3D



Il Modello del Ray Tracing

- La crescente complessità delle scene e l'alto parallelismo a livello di dati trovano la loro massima espressione nel **Ray Tracing** - Tecnica di rendering che simula il comportamento fisico della luce
- Ogni pixel richiede il calcolo di più raggi luminosi e delle loro interazioni (**riflessioni, rifrazioni, ombre**).
- Ogni pixel potrebbe essere calcolato **simultaneamente**, con sufficiente parallelismo



https://www.youtube.com/watch?v=EQ_4I2HIXCc



Lunar Landing: NVIDIA RTX Real-Time Ray Tracing Demo

RTX
OFF



RTX
ON



Evoluzione dei Graphics Accelerator negli anni '90

Graphics Accelerator

- Hardware specializzato progettato per **accelerare le operazioni grafiche**, riducendo il carico sulla CPU.
- **Perché furono introdotti?**
 - Crescente domanda di grafica più complessa e fluida
 - CPU insufficienti per gestire carichi grafici avanzati, specialmente in 3D
 - Necessità di migliorare le prestazioni nei videogiochi e nelle applicazioni grafiche.
- **Caratteristiche chiave**
 - Accelerazione 2D hardware (es. blitting, riempimento aree)
 - Verso fine '90: primi passi nell'accelerazione di funzioni 3D basilari (texture mapping, shading, etc.)
 - Memoria dedicata per frame buffer



S3 ViRGE (1995)



3dfx Voodoo (1996)



NVIDIA RIVA 128 (1997)

Limiti dell'Era Pre-GPU: La Sfida dell'Accelerazione 3D

Dall'Accelerazione 2D al 3D

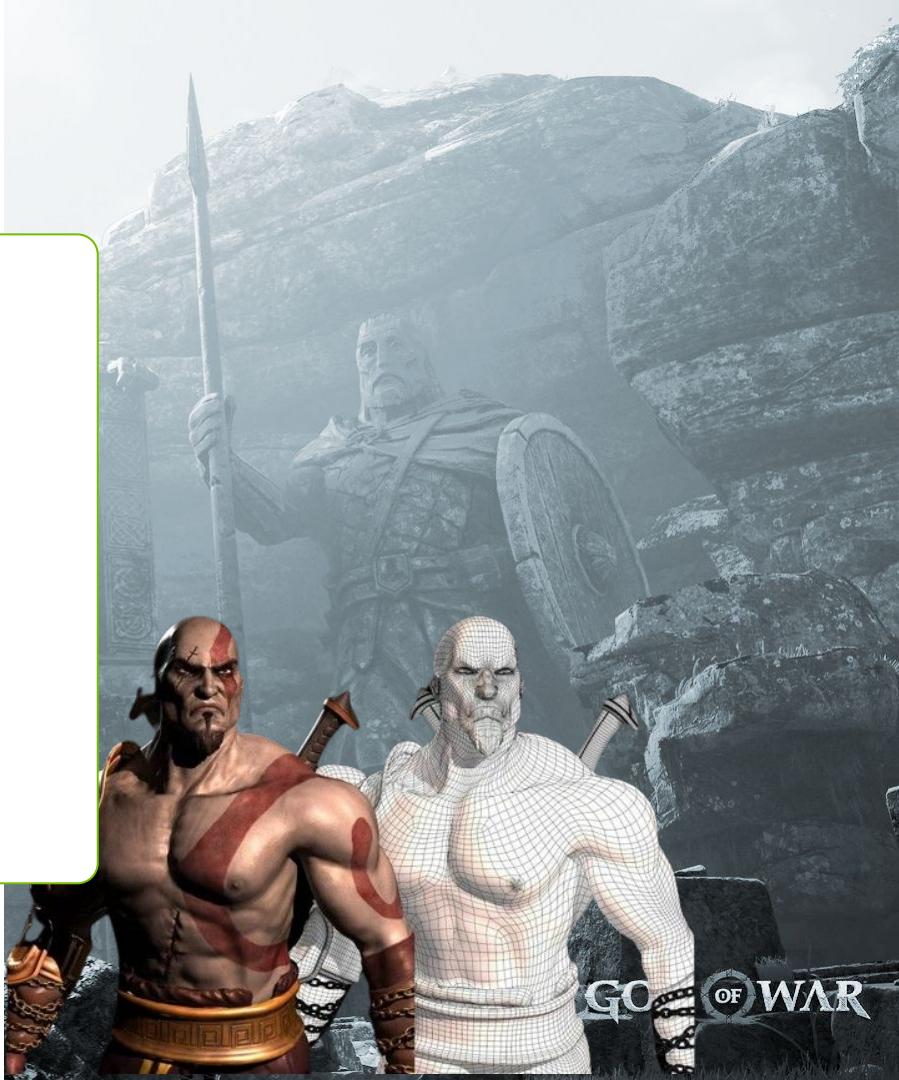
- I Graphics Accelerator degli anni '90 introdussero le prime funzionalità 3D, ma il passaggio non fu privo di sfide.

Principali Limitazioni

- **Prestazioni insufficienti**
 - La CPU era ancora sovraccarica nel gestire operazioni 3D complesse
 - Esempio: basso frame rate in scene con molti poligoni e texture (<15 FPS)
- **Flessibilità Limitata**
 - L'hardware non era ottimizzato per le nuove funzioni 3D
 - Difficoltà nell'implementare algoritmi avanzati di shading e altre tecniche di rendering.
- **Mancanza di Programmabilità**
 - Le pipeline grafiche dell'epoca erano **fisse** (fixed-function), cioè predefinite dall'hardware e non modificabili dai programmati.
 - Questo limitava la capacità di creare nuove tecniche di rendering, bloccando l'innovazione.

Videogiochi Moderni

- **Crescente Complessità delle Scene**
 - Centinaia di migliaia di triangoli per oggetto
 - Milioni di pixel da elaborare per frame
 - Illuminazione dinamica, particelle, fisica realistica
- **Evoluzione Annuale**
 - Grafica sempre più sofisticata e realistica
 - Richiesta di prestazioni in costante aumento
 - Mondi di gioco più vasti e dettagliati
- **Caratteristiche del Carico di Lavoro**
 - Enorme quantità di calcoli per frame
 - Operazioni per lo più indipendenti tra loro
 - Alto parallelismo a livello di dati



Spinte alla Tecnologia GPU

- Le esigenze crescenti nei diversi settori hanno superato le capacità delle CPU tradizionali, creando una domanda per hardware più specializzato come le GPU.

- **Industria dei Videogiochi**

- **Esigenza:** Calcoli paralleli per rendering 3D complesso e interattivo.
- **Necessità:** Hardware capace di gestire simultaneamente numerosi elementi grafici ed effetti.

- **Industria Cinematografica**

- **Esigenza:** Elaborazione parallela per effetti visivi e simulazioni realistiche.
- **Necessità:** Hardware che accelera rendering e simulazioni di grandi quantità di dati.

- **Visualizzazione Scientifica**

- **Esigenza:** Gestione e visualizzazione di grandi set di dati scientifici.
- **Necessità:** Hardware per rendering 3D complessi e simulazioni in tempo reale.

- **Industria Medica**

- **Esigenza:** Ricostruzione e visualizzazione di immagini 3D da TAC e RMN.
- **Necessità:** Hardware per elaborazione rapida per rendering volumetrico e analisi di immagini.

- ...

Introduzione alle GPU

- L'evoluzione delle tecnologie grafiche ha portato alla creazione delle **GPU (Graphics Processing Units)**, segnando un punto di svolta nell'elaborazione grafica e nel calcolo parallelo.

Cosa è una GPU?

- Hardware progettato per l'elaborazione parallela, ottimizzato per gestire e accelerare il rendering grafico e le operazioni di calcolo intensivo.*

Funzioni Principali

- Rendering grafico**
 - Trasformazione di modelli 3D in immagini 2D
 - Calcolo di illuminazione, ombreggiatura e effetti visivi in tempo reale
- Calcolo Parallelo Massivo**
 - Esecuzione simultanea di migliaia di operazioni
 - Ottimizzazione per task che richiedono calcoli ripetitivi su grandi set di dati



Nvidia GeForce 256

“the world's first 'GPU'” (1999)

L'Evoluzione verso il GPGPU

Dalla Grafica al Calcolo Generale

- Le GPU, nate per l'accelerazione grafica, mostravano un potenziale non sfruttato per altri tipi di calcolo.
- **Intuizione chiave:** La struttura parallela delle GPU poteva essere applicata a problemi al di fuori del dominio grafico (applicazioni-general purpose), come il rendering scientifico o simulazioni fisiche.

Introduzione al GPGPU

- **GPGPU** (General-Purpose computing on Graphics Processing Units) rappresenta l'utilizzo delle GPU per calcoli non grafici.
- **Primi approcci:** Complessi e limitati, richiedevano di "ingannare" la GPU utilizzando API grafiche come OpenGL o DirectX per presentare i calcoli come operazioni grafiche.
- **Sfide iniziali:** Programmazione complessa, limiti nell'uso di dati in virgola mobile, mancanza di strumenti di debug e profiling.
- **Sviluppo di framework dedicati:** Con la crescente domanda di calcolo general-purpose su GPU, sono stati sviluppati framework e linguaggi specifici come CUDA (NVIDIA) e OpenCL, che hanno semplificato notevolmente la programmazione GPGPU.
- **Applicazioni odierne:** Crittografia, Analisi finanziaria, Fluidodinamica computazionale, Machine Learning / Deep Learning, etc.

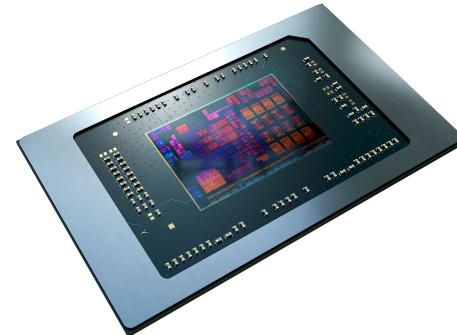
GPU Dedicated vs. Integrate

- La classificazione delle GPU si basa principalmente su due architetture distinte: **Dedicate** e **Integrate**



GPU Dedicata

- Prestazioni elevate
- Memoria video (VRAM) separata.
- Consuma più energia
- Costosa
- Ideale per gaming e lavori grafici intensivi
- Utilizzata principalmente in PC da gaming, workstation, server.



GPU Integrata (iGPU)

- Prestazioni moderate
- Condivide la memoria di sistema
- Consuma meno energia
- Economica (inclusa nel processore)
- Sufficiente per attività quotidiane (streaming video, navigazione web)
- Comune in laptop, ultrabook, dispositivi mobili.

Evoluzione e Impatto delle GPU nel Gaming Moderno

- Le moderne console utilizzano un **SoC** (System on Chip) che integra la CPU e la GPU in un unico chip.
- Questa GPU è **dedicata ma integrata nel chip** con la CPU, condividendo la memoria unificata (RAM).



Sony Playstation 5



XBOX Series X



OneXPlayer



Asus ROG Ally



Nintendo Switch



Steam Deck

Principali Aziende di GPU Oggi



NVIDIA

Leader nel mercato HPC, AI e gaming
Tecnologie: CUDA, Tensor Cores



Qualcomm

Leader nelle GPU per dispositivi mobili
Tecnologie: Adreno, Snapdragon SoCs



AMD

Secondo maggior produttore di GPU
Tecnologie: ROCm, RDNA



Intel

Nuovo entrato nel mercato GPU
Tecnologie: OneAPI, Xe Architecture

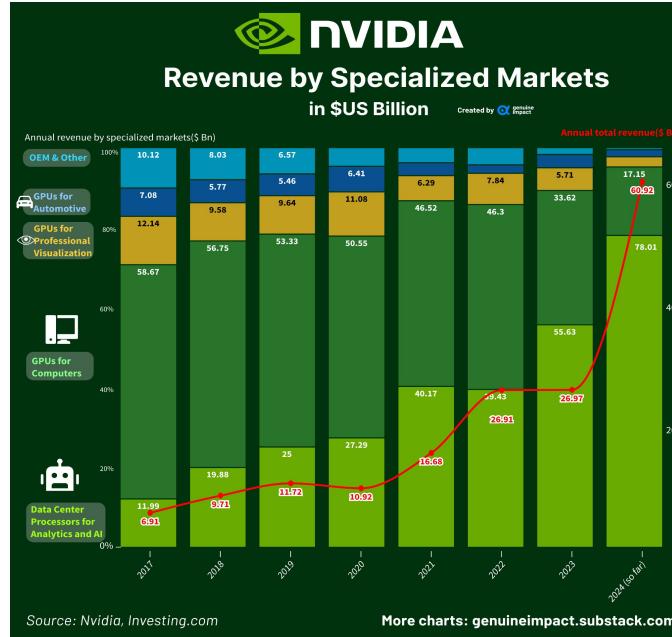
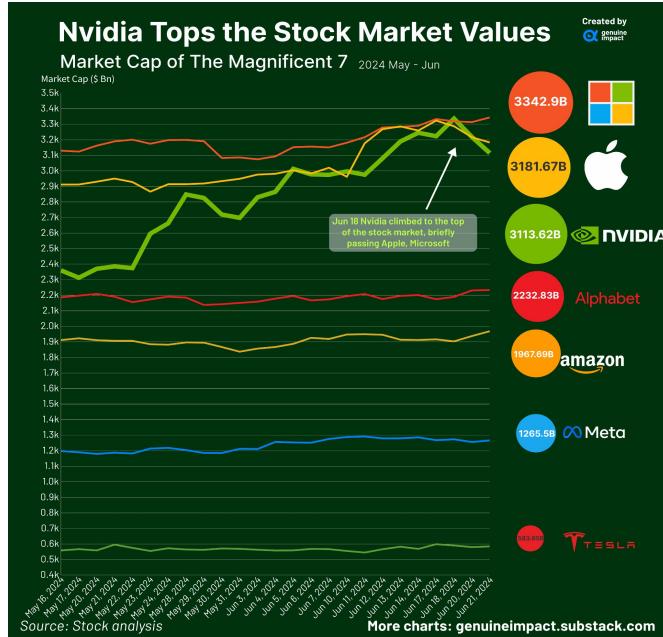


Apple

GPU integrate nei chip della serie M
Tecnologia: Metal API

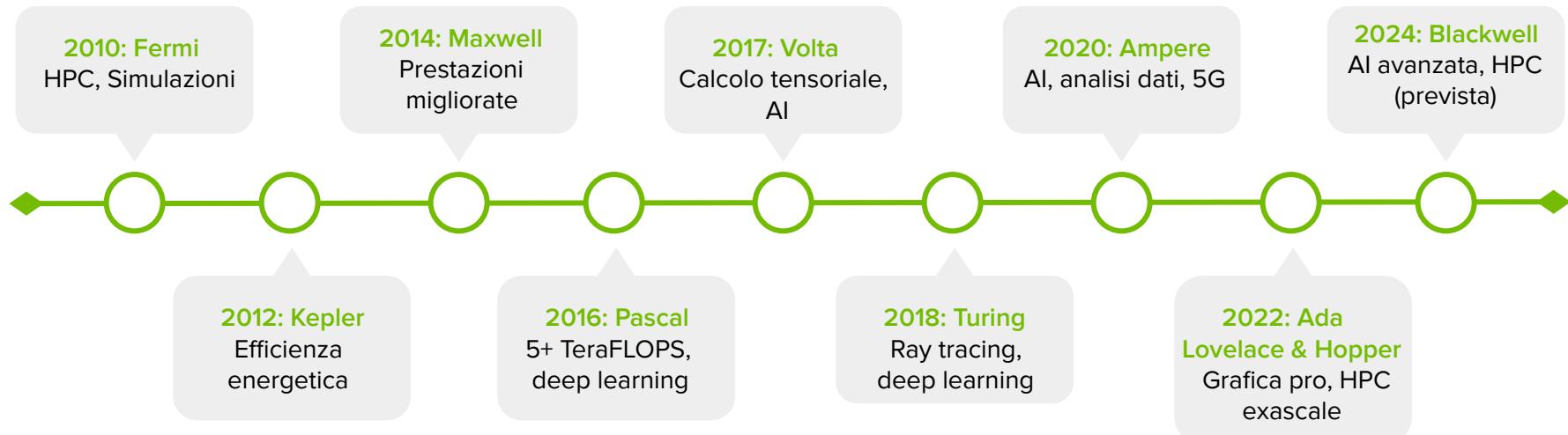
NVIDIA: Dominio di Mercato e Crescita dei Ricavi

- Capitalizzazione di Mercato:** NVIDIA ha superato brevemente Apple e Microsoft, raggiungendo la vetta della capitalizzazione di mercato tra le aziende tecnologiche a giugno 2024.
- Crescita dei Ricavi per Settore:** I ricavi di NVIDIA nel settore dei data center per AI e analytics sono aumentati drasticamente, passando dall'11.99% nel 2017 a 75.01% 2024.



Evoluzione delle Architetture GPU NVIDIA

- **Progressione Tecnologica:** Da Fermi a Blackwell, ogni generazione ha portato significativi avanzamenti nelle capacità di calcolo e nell'efficienza energetica.
- **Adattamento al Mercato:** L'evoluzione riflette il passaggio da un focus su grafica e HPC a un'enfasi crescente su AI, deep learning e calcolo ad alte prestazioni.



Dominio di NVIDIA nell'AI Generativa

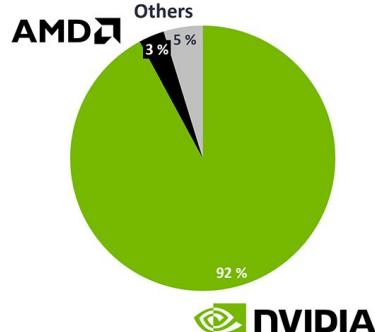
- NVIDIA possiede il 92% del mercato delle GPU per data center, cruciali per le applicazioni di AI generativa.
- Aumento del 272% delle entrate trimestrali, passando da \$4.3 miliardi a \$16 miliardi previsti.



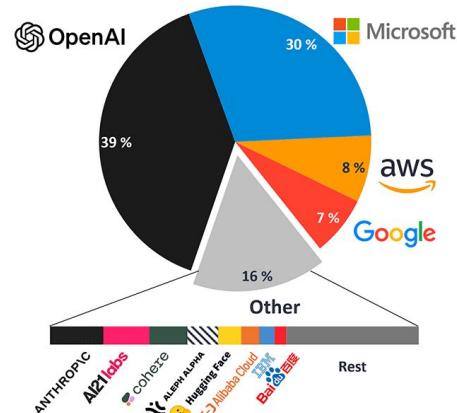
December 2023

Generative AI: Market share of leading vendors 2023

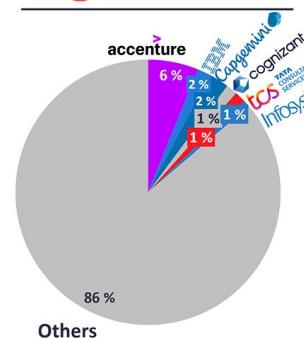
1 Data center GPUs



2 Models & platforms



3 Services



Note: Numbers are rounded and might not add up to 100%; Market share is based on 2023 market sizes (based on revenue).

Source: IoT Analytics Research 2023-Generative AI Market Report 2023-2030. We welcome republishing of images but ask for source citation with a link to the original post and company website.

AI Generativa - Large Language Models (LLMs)



Prompt

Write a bit for my standup show about turning 30

Gemini Ultra ⓘ
Google - \$20/mo

33%

GPT-4

OpenAI - \$20/mo

67%

Claude 3 ⓘ

Anthropic - \$20/mo



You know, they say 30's the new 20. But who are "they"? Some seriously delusional people that's who! I woke up on my 30th birthday, and it wasn't new shoes and parties I was greeted with – it was a dawning sense of my own mortality. Is this it? Does everything just go downhill from here? Please tell me someone has an instruction manual for adulthood. ...

Ladies and gentlemen, I recently hit a milestone in my life. That's right, I turned 30. Now, turning 30 is a lot like logging into an old email account. You're excited to see what's there, but also terrified of the reminders of past decisions. "Why did I sign up for a newsletter on how to make my own soap?" ...

"So, I recently turned 30, and let me tell you, it's been a wild ride. It's like I woke up one day, and suddenly, everything started making this weird cracking sound... turns out, it was just my joints.

But hey, turning 30 isn't all bad. I mean, I finally feel like I'm being taken seriously. When I was in my 20s, people would always ask me, 'When are you going to get a real job?' Now, they ask me, 'When are you going to settle down and have kids?' I can't win! ...



Alcuni dettagli:

- **Modelli:** ChatGPT, Gemini, Claude, Grok, Perplexity, Bard, LLaMA, Mistral, etc
- **Dataset:** Decine di trilioni di tokens (parole e frasi provenienti da *archivi web e contenuti pubblici*)
- **(Esempio) ChatGPT-4:** ~1,8 trilioni di parametri (oltre 10 volte più di GPT-3)
 - **Infrastruttura:** Cluster di 25.000 GPU NVIDIA A100 (cloud Azure di Microsoft)
 - **Tempo di training:** 90-100 giorni. Oltre 1 milione di GPU ore (circa 41.667 giorni su una singola GPU)
 - **Consumo energetico:** 2,5 milioni di kWh (equivalenti a 170-250 case per un anno)



AI Generativa - Large Language Models (LLMs)



Prompt

Write a bit for my standup show about turning 30

Gemini Ultra ⓘ
Google - \$20/mo

33%

GPT-4 ⓘ
OpenAI - \$20/mo

67%

Claude 3 ⓘ
Anthropic - \$20/mo

Gemini

MISTRAL
AI_

You know, they say 30's the new 20. But who are "they"? Some seriously delusional people that's who! I woke up on my 30th birthday, and it wasn't new shoes and parties I was greeted with – it was a dawning sense of my own mortality. Is this it? Does everything just go downhill from here? Please tell me someone has an instruction manual for adulthood. ⏱

Ladies and gentlemen, I recently hit a milestone in my life. That's right, I turned 30. Now, turning 30 is a lot like logging into an old email account. You're excited to see what's there, but also terrified of the reminders of past decisions. "Why did I sign up for a newsletter on how to make my own soap?" ⏱

"So, I recently turned 30, and let me tell you, it's been a wild ride. It's like I woke up one day, and suddenly, everything started making this weird cracking sound... turns out, it was just my joints.

But hey, turning 30 isn't all bad. I mean, I finally feel like I'm being taken seriously. When I was in my 20s, people would always ask me, 'When are you going to get a real job?' Now, they ask me, 'When are you going to settle down and have kids?' I can't win! ⏱



Curiosità sul (futuro) modello Grok 3:

- GPU Utilizzate:** 100.000 NVIDIA H100 (~40.000 dollari ciascuna)
- Costo Totale (Stimato):** 3 miliardi di dollari per l'addestramento
- Tempistiche di Addestramento:** 22 Luglio 2024 - Dicembre 2024
- Cluster di Addestramento:** Considerato come il "più potente cluster di addestramento IA del mondo"
- Consumo Stimato:** Il consumo energetico totale di Grok 3 si stima sia equivalente a quello necessario per alimentare circa 10.710 case per un anno intero (128,52 milioni di kWh)

LLaMA
by Meta

AI Generativa - Creazione di Immagini da Testo



Teddy bears swimming at the Olympics 400m Butterfly event.



A cute corgi lives in a house made out of sushi.



A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.



A fluffy baby sloth with a knitted hat trying to figure out a laptop, close up.



A sheep in a wine glass.



A futuristic city with flying cars.



A large array of colorful cupcakes, arranged on a maple table to spell MUSE.



A brain riding a rocketship heading towards the moon. A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.



A strawberry mug filled with white sesame seeds. The mug is floating in a dark chocolate sea.



Manhattan skyline made of bread.



Astronauts kicking a football in front of Eiffel tower.



Two cats doing research.



3D mesh of Titanic floating on a water lily pond in the style of Monet.

Alcuni dettagli:

- **Modelli:** Midjourney, Stable Diffusion, DALL-E 2, etc
- **Dataset:** Miliardi di immagini e descrizioni testuali
- **(Esempio) Stable Diffusion:** Addestrato su miliardi di coppie immagine-testo (provenienti da [LAION](#))
 - **Infrastruttura:** 256 GPU NVIDIA A100 (80 GB ciascuna)
 - **Tempo di training:** 150.000 GPU ore (6.250 giorni su una GPU)

AI Generativa - Creazione di Video da Testo



"Lion with eagle wings in desert landscape"



"Rabbit hopping with boxing gloves, cuts to tiger wearing boxing gloves in ring"

Alcuni dettagli:

- **Modelli:** Meta Make-A-Video, Runway, Sora, Kling
- **Dataset:** Milioni di video e descrizioni testuali
- **(Esempio) Meta Make-A-Video:** Decine di miliardi di parametri
 - **Infrastruttura:** Cluster di 2.000 GPU NVIDIA A100
 - **Tempo di training:** Oltre 500.000 GPU ore (circa 20.833 giorni su una singola GPU)
 - **Consumo energetico:** Stimato oltre 5 milioni di kWh (equivalenti al fabbisogno di 400 case per un anno)

Altre Applicazioni di Deep Learning

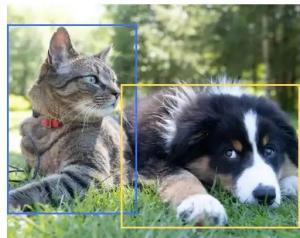
Image Classification



Classification
Cat

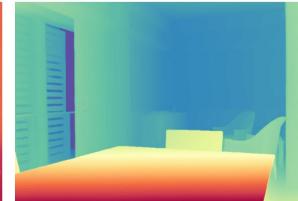


Classification, Localization
Cat



Object Detection
Cat, Dog

Single Image Depth Estimation



Semantic Segmentation

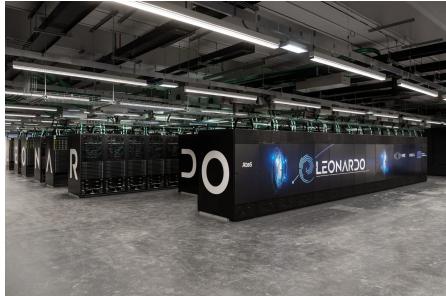


Super-Resolution / 3D / Novel View Synthesis / etc



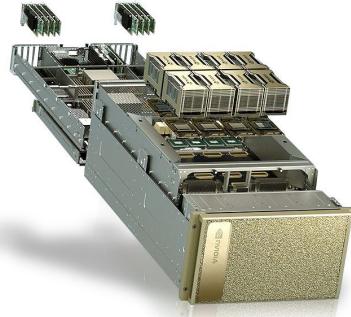
Esempi di Sistemi GPU

- Le GPU, con la loro potenza di calcolo parallelo, sono il motore di una vasta gamma di sistemi:



Supercomputer (es. Leonardo)

HPC, Calcolo parallelo massivo, AI su larga scala
Costo: \$100M - \$600M+



Server GPU

AI avanzata, analisi dati massive
Costo: \$200K - \$500K



Workstation Professionale

Rendering 3D, editing video 4K/8K
Costo: \$5K - \$40K



Desktop Gaming

Gaming ad alte prestazioni, rendering
Costo: \$1.5K - \$5K



GPU Mobile

Grafica e AI su dispositivi mobili
Costo: \$400 - \$2.5K



Embedded GPU

Edge computing, IoT avanzato, AI per robot
Costo: \$500 - \$5K

Esempi di Sistemi GPU

- Panoramica delle GPU NVIDIA più recenti e comunemente utilizzate per ciascun sistema



NVIDIA H100

Architettura Hopper; HPC, AI su larga scala
Costo: \$35K - \$42K



NVIDIA A100

Architettura Ampere; AI avanzata
Costo: \$12K - \$15K



NVIDIA RTX 6000

Architettura Ada; Rendering 3D, editing video 4K/8K
Costo: \$6.5K - \$8K



NVIDIA GeForce RTX 4090
Architettura Ada; Gaming, rendering
Costo: \$1.6K - \$2K



NVIDIA GeForce RTX 4080 Laptop GPU
Architettura Ada; Grafica e AI su dispositivi mobili
Costo: \$1.2 - \$2.5K



NVIDIA Jetson Xavier NX
Architettura Volta; Edge computing, IoT avanzato
Costo: \$350-500

Evoluzione della Potenza di Calcolo

- In soli due decenni, una singola GPU ha raggiunto una **potenza di calcolo quasi tripla** rispetto al supercomputer più potente del 2001.



NVIDIA H100 SXM (2023)

FP32/64 CUDA Core: 16.896/8448

Peak FP64 TFLOP/s: 34

Potenza di Consumo (TDP): 700W



ASCI White, SP Power3 375 MHz, IBM
Lawrence Livermore National Laboratory

CPU: 8.192

Peak FP64 TFLOP/s: 12.3

Consumo Energetico: 6MW

Differenze tra CPU e GPU

Differenze tra CPU e GPU - Utilizzo



CPU (Central Processing Unit)

- Esegue il **sistema operativo** e la maggior parte dei **programmi generali** (Navigazione web, multitasking quotidiano, etc)
- Adatta per compiti che richiedono **calcoli complessi** e **operazioni logiche**.

GPU (Graphics Processing Unit)

- Inizialmente progettata per il **rendering grafico** (videogiochi, modellazione 3D, etc).
- Ora utilizzata anche per applicazioni di **calcolo parallelo** come l'apprendimento automatico, la simulazione scientifica, e la crittografia.

Differenze tra CPU e GPU - Architettura



CPU (Central Processing Unit)

- **Core:** Da 4-16 core per consumer CPU, fino a 128 core per CPU server di fascia alta.
- **Frequenza:** Tipicamente tra 2.5 GHz e 5 GHz.
- **Gestione delle Istruzioni:** Predizione delle diramazioni (*branch prediction*), esecuzione fuori ordine (*out-of-order execution*), lunga *pipeline* di elaborazione

GPU (Graphics Processing Unit)

- **Core:** Da centinaia a migliaia di core (es. Nvidia RTX 4090 ha 16384 CUDA cores)
- **Frequenza:** Tipicamente tra 1 GHz e 2 GHz.
- **Architettura parallela:** Design SIMD (Single Instruction, Multiple Threads) per NVIDIA GPUs.

Differenze tra CPU e GPU - Memoria



CPU (Central Processing Unit)

- **Tipo di memoria:** DDR4 o DDR5.
- **Larghezza di banda:** Fino a 100 GB/s con DDR5.
- **Cache:** Cache di grandi dimensioni per core (fino a 72 MB di L3 cache).
- **Accesso alla memoria:** Accesso diretto e condiviso con il sistema operativo e altre applicazioni.

GPU (Graphics Processing Unit)

- **Tipo di memoria:** GDDR6, GDDR6X, HBM2.
- **Larghezza di banda:** Fino a 1000 GB/s con HBM2.
- **Memoria dedicata:** Memoria dedicata solo alla GPU, separata dalla RAM del sistema.
- **Dimensione:** 4-24 GB per GPU di consumo, fino a 48 GB o più per GPU professionali.

Differenze tra CPU e GPU - Prestazioni



CPU (Central Processing Unit)

- Maggiore velocità per singolo thread (**bassa latenza**).
- Eccellente per carichi di lavoro **sequenziali**.
- Migliore gestione di flussi di controllo complessi e operazioni logiche.
- **Bassa latenza** per operazioni singole e accesso alla memoria.

GPU (Graphics Processing Unit)

- Maggiore capacità di calcolo parallelo massivo.
- Eccellente per carichi di lavoro altamente **parallelizzabili** (es. rendering grafico, machine learning).
- **Throughput superiore** per operazioni numeriche semplici e ripetitive.

Differenze tra CPU e GPU - Svantaggi



CPU (Central Processing Unit)

- **Parallelismo limitato:** Meno efficiente per operazioni altamente parallele.
- **Scalabilità:** Difficoltà nell'aumentare il numero di core (vincoli termici e di consumo energetico).
- **Costo/prestazione:** Costose per ottenere alte prestazioni in task paralleli.

GPU (Graphics Processing Unit)

- **Versatilità limitata:** Non adatta a tutti i tipi di carichi di lavoro.
- **Latenza:** Più alta per operazioni singole rispetto alla CPU.
- **Complessità di programmazione:** Richiede competenze specifiche per l'ottimizzazione.
- **Consumo energetico:** Elevato sotto carichi intensivi.

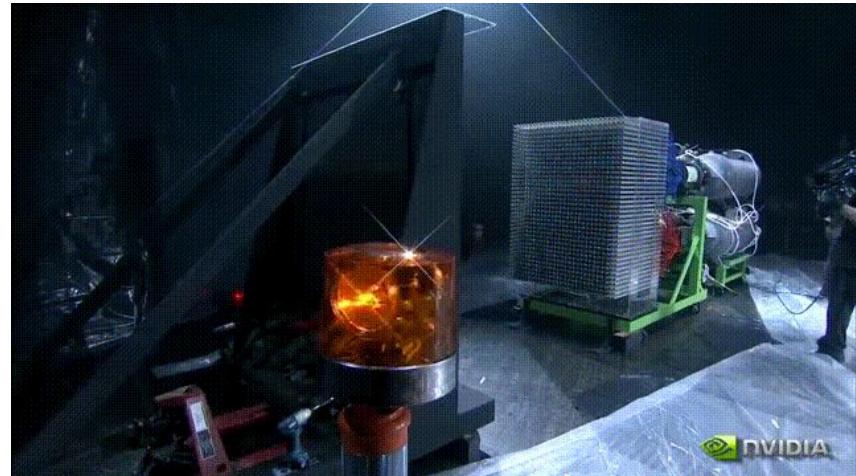
Efficienza nel Rendering: CPU vs. GPU

- Come dimostrato nel video NVIDIA (2008), una GPU può generare l'immagine della Monna Lisa in pochi millisecondi, mentre una CPU impiega molto più tempo, eseguendo compiti in modo sequenziale.

<https://www.youtube.com/watch?v=QkHbhzP8-ww>



CPU



GPU

CPU vs GPU di Ultima Generazione - Esempio

Caratteristica	AMD Ryzen 9 7950X (CPU)	NVIDIA GeForce RTX 4090 (GPU)
Architettura	Zen 4	Ada Lovelace
Numero di Core	16 core / 32 thread	16,384 CUDA core
Frequenza Base	4.5 GHz	2.23 GHz
Frequenza Boost	Fino a 5.7 GHz	Fino a 2.52 GHz
Cache	80 MB (L2 + L3)	72 MB L2
Memoria Supportata	DDR5-5200	24 GB GDDR6X
Larghezza di Banda	~100 GB/s (con DDR5)	1,008 GB/s
TDP (Thermal Design Power)	170W	450W
Transistor	~6.5 miliardi	76.3 miliardi
Prezzo di Lancio	\$699	\$1,599
Prestazioni FP32	~1 TFLOPS	82.6 TFLOPS

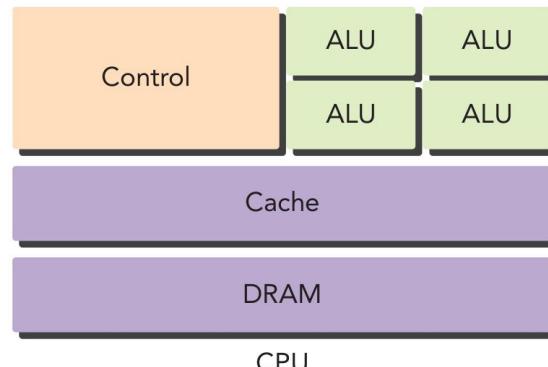
- **Verde:** Vantaggio significativo
- **Arancione:** Svantaggio relativo
- **Nero:** Aspetto Neutro

Perché limitarsi a un solo tipo di processore quando possiamo sfruttare i punti di forza di entrambi?

Architetture Eterogenee

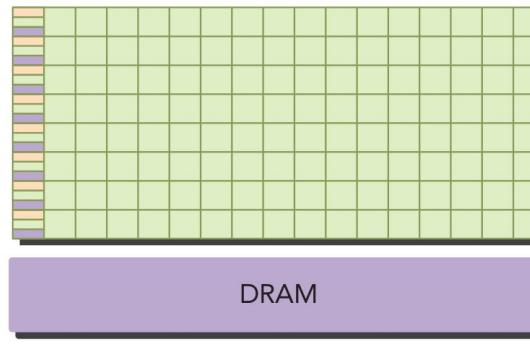
Architetture Eterogenee

- **Cosa è una Architettura Eterogenea?** Un'architettura eterogenea è una struttura di sistema che integra diversi tipi di processori o core di elaborazione all'interno dello stesso computer o dispositivo.
- **Ruoli**
 - **CPU (Host)**: gestisce l'ambiente, il codice e i dati
 - **GPU (Device)**: co-processore, accelera calcoli intensivi (hardware accelerator)
- **Connessione**: GPU collegate alla CPU tramite bus **PCI-Express**
- **Struttura del Software**: Applicazioni divise in **codice host** (CPU) e **device** (GPU)



Riduce la Latenza

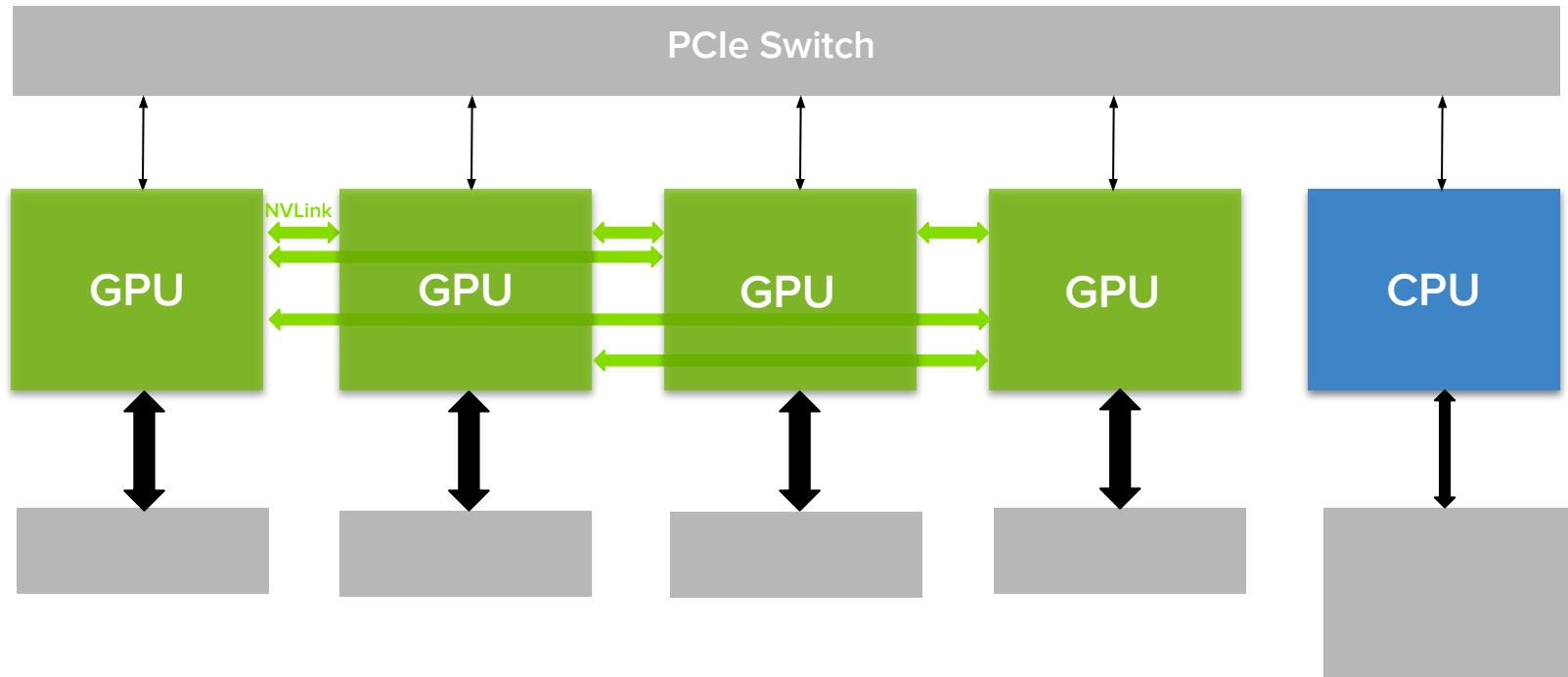
PCIe Bus



Massimizza il Throughput

Sistemi Multi-GPU in Architetture Eterogenee

- **Cos'è un Sistema Multi-GPU?** Un sistema che integra più GPU all'interno della stessa architettura eterogenea, permettendo di distribuire e parallelizzare il carico di lavoro su diverse unità di elaborazione grafica.



Architetture Eterogenee

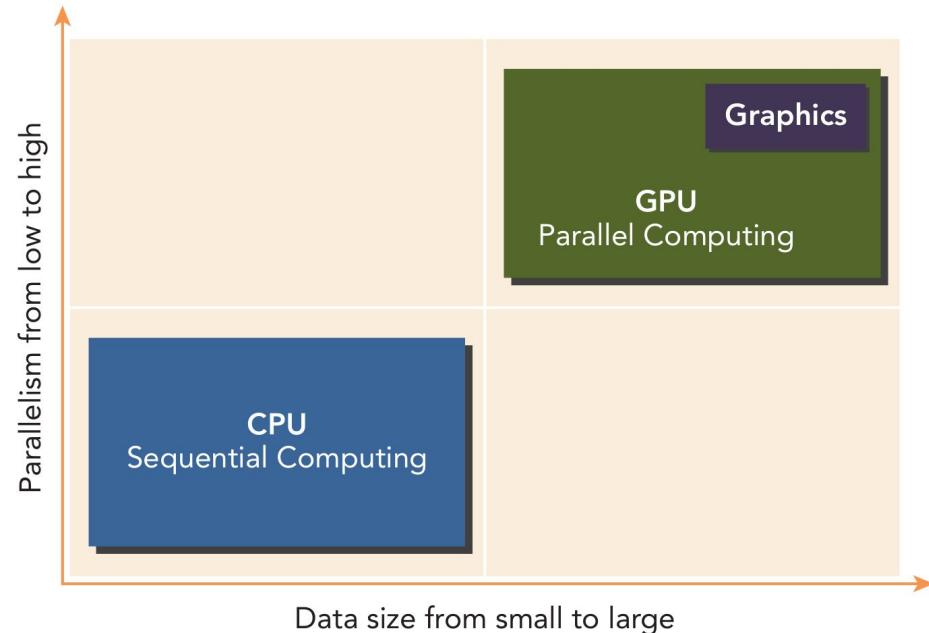
- Due dimensioni che differenziano lo scope di applicabilità di CPU e GPU

1. Livello di parallelismo:

- CPU: Basso (pochi thread)
- GPU: Alto (migliaia di thread)

2. Dimensione dei dati:

- CPU: Piccola-Media (KB-MB)
- GPU: Grande-Massiva (GB-TB)



Architetture Eterogenee

- Due dimensioni che differenziano lo scope di applicabilità di CPU e GPU

1. Livello di parallelismo:

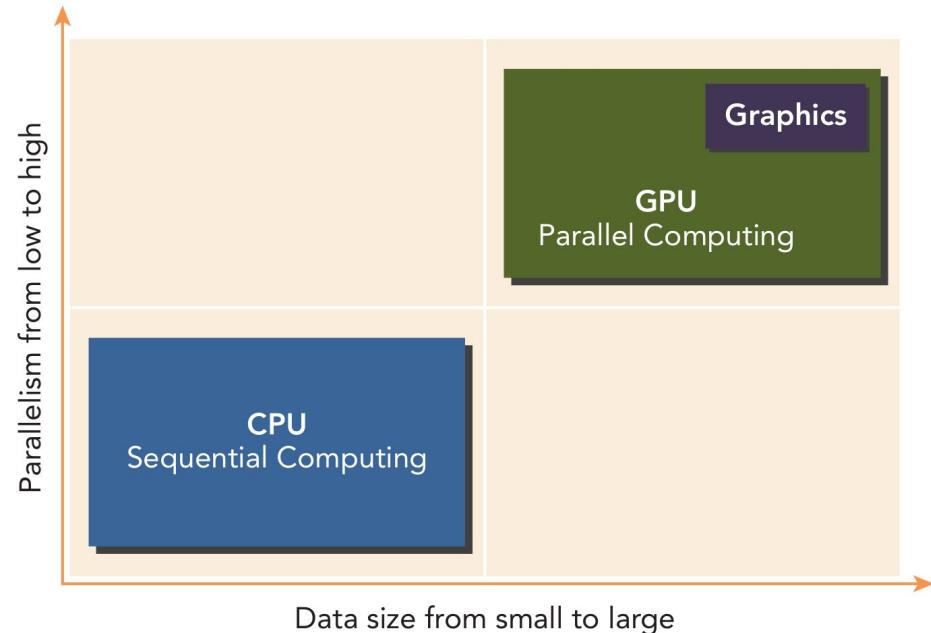
- CPU: Basso (pochi thread)
- GPU: Alto (migliaia di thread)

2. Dimensione dei dati:

- CPU: Piccola-Media (KB-MB)
- GPU: Grande-Massiva (GB-TB)

Approccio ottimale:

- Combinare CPU e GPU per massimizzare le prestazioni
- Basso parallelismo + Dati limitati → CPU
- Alto parallelismo + Dati massicci → GPU



Architetture Eterogenee - Struttura del Software

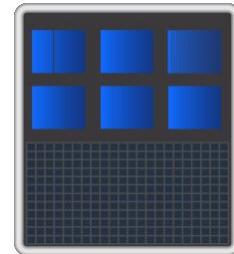
Application Code



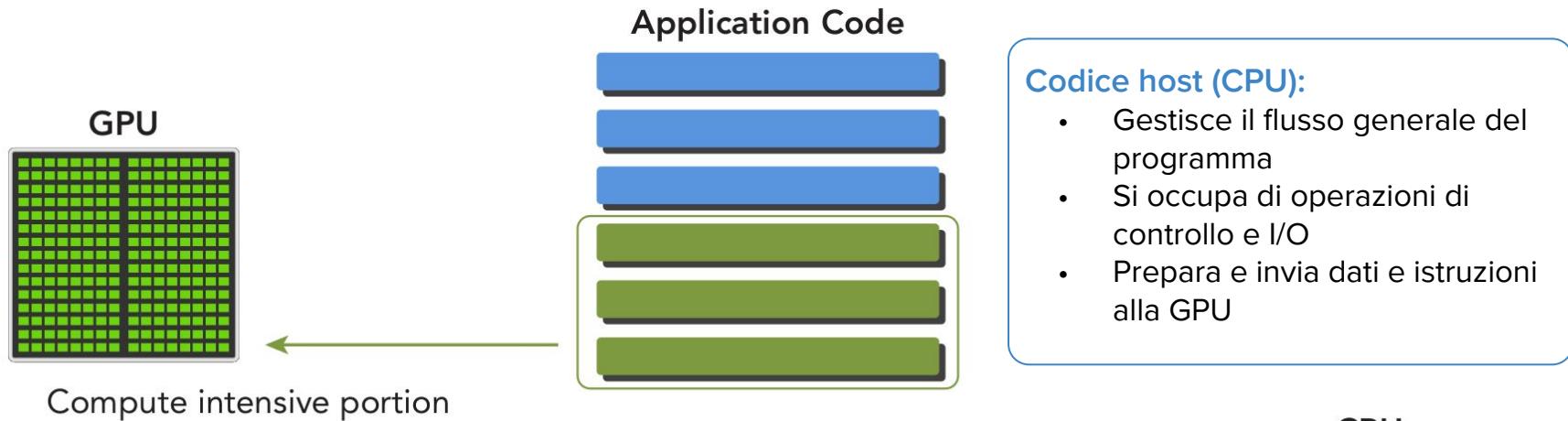
Codice host (CPU):

- Gestisce il flusso generale del programma
- Si occupa di operazioni di controllo e I/O
- Prepara e invia dati e istruzioni alla GPU

CPU

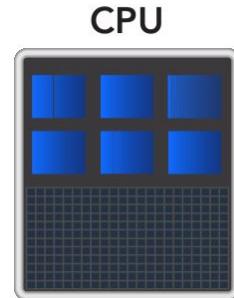


Architetture Eterogenee - Struttura del Software

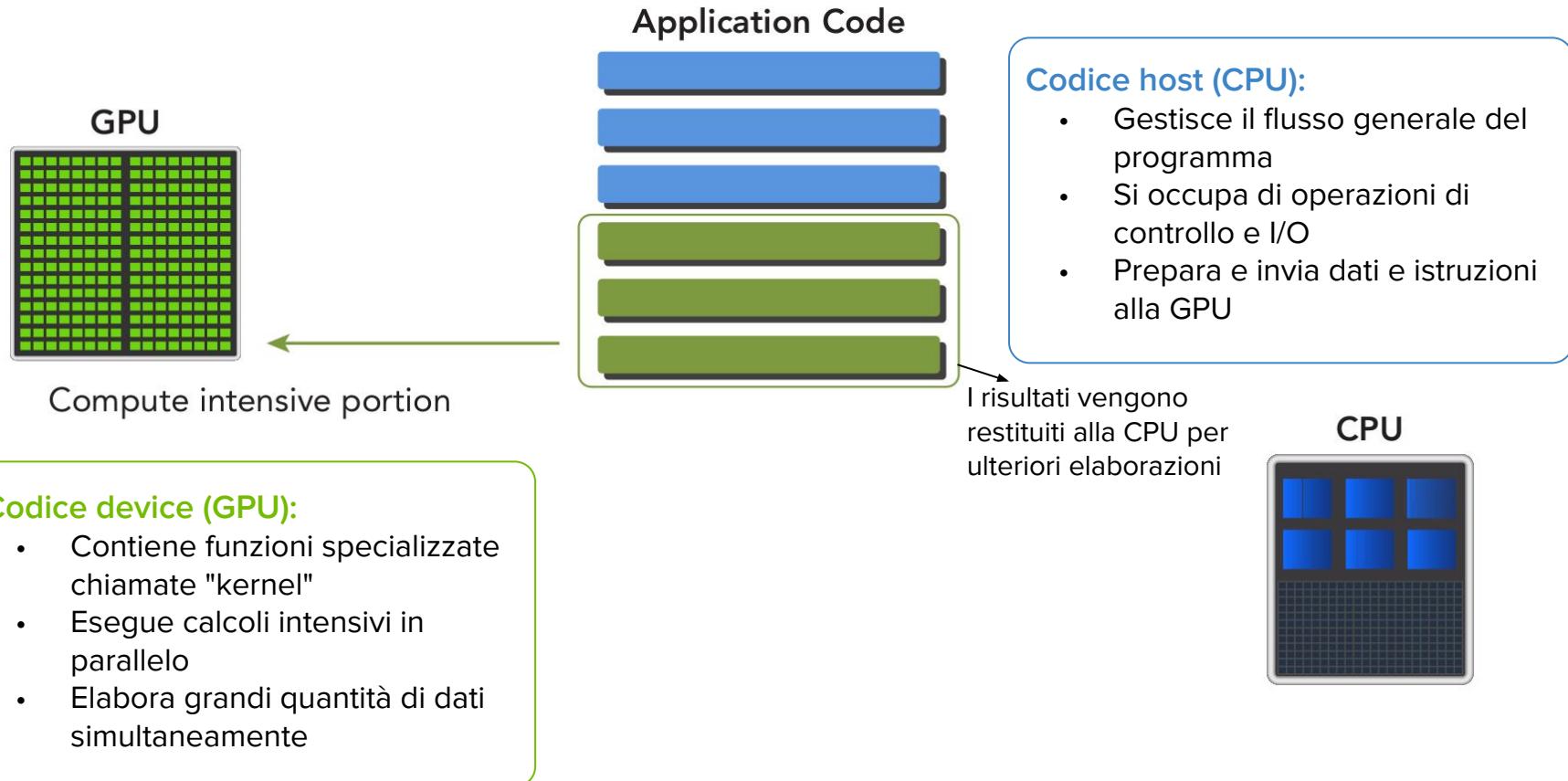


Codice device (GPU):

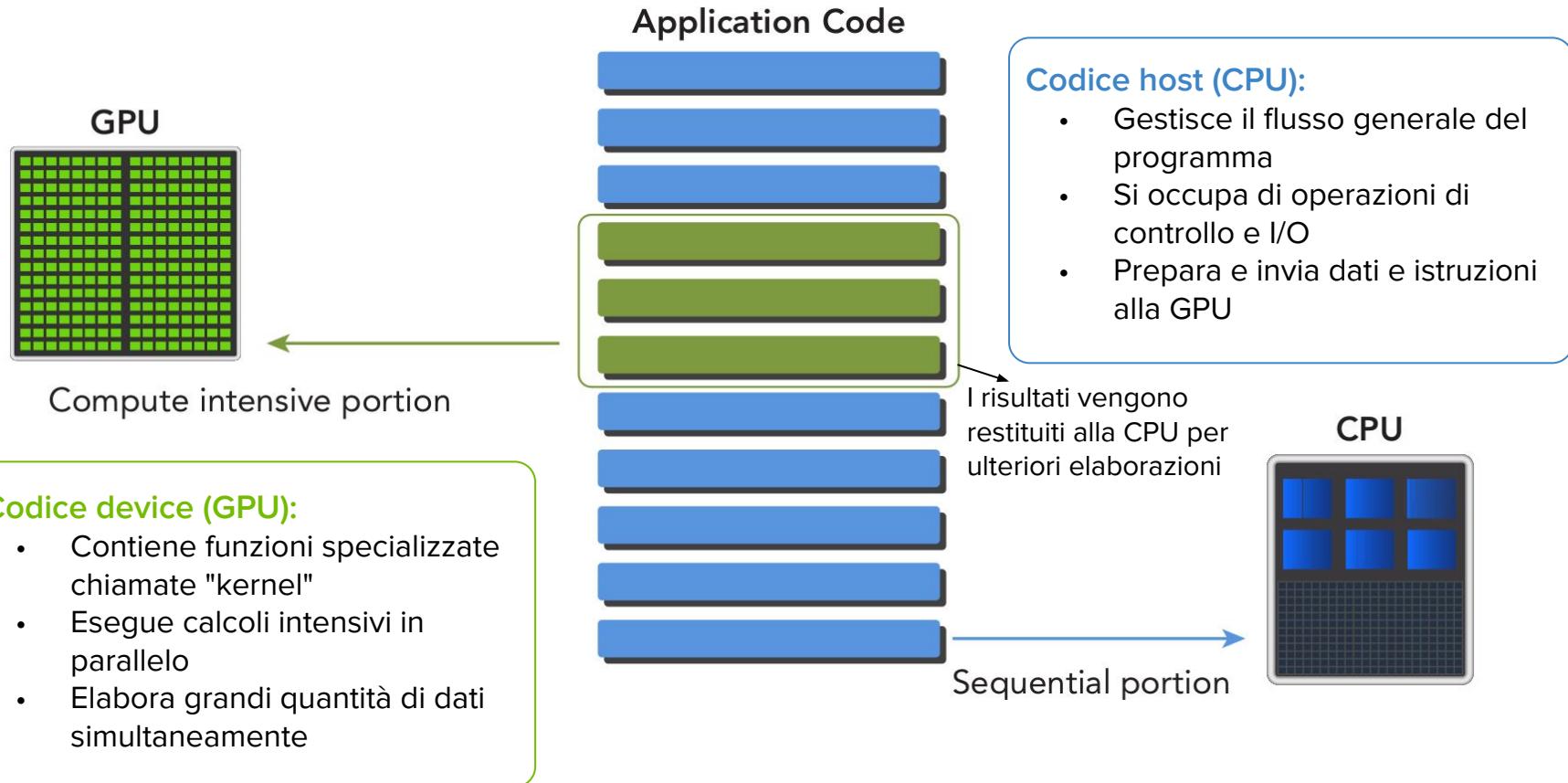
- Contiene funzioni specializzate chiamate "kernel"
- Esegue calcoli intensivi in parallelo
- Elabora grandi quantità di dati simultaneamente



Architetture Eterogenee - Struttura del Software



Architetture Eterogenee - Struttura del Software



Come possiamo supportare l'esecuzione congiunta di CPU e GPU in un'applicazione?

CUDA

Un piattaforma per il calcolo parallelo

CUDA: Piattaforma di Calcolo Parallelo

Cos'è CUDA (Compute Unified Device Architecture)?

- Piattaforma di calcolo parallelo general-purpose ideato da NVIDIA (lanciata nel 2007)
- Modello di programmazione per GPU NVIDIA

Obiettivo

- Sfruttare la potenza di calcolo parallelo delle GPU
- Semplificare lo sviluppo per sistemi CPU-GPU

CUDA come Ecosistema

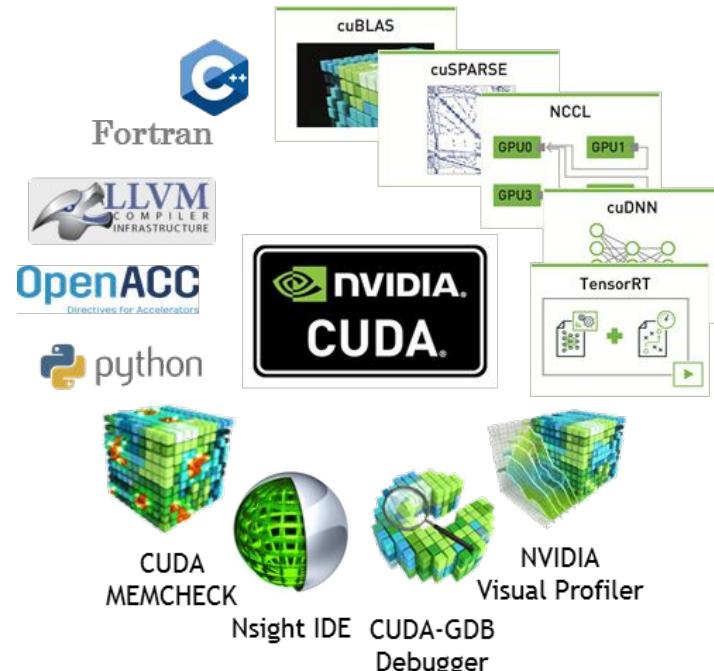
- Compilatore (nvcc), Profiler/Debugger (Nsight), Librerie Ottimizzate, Strumenti di Sviluppo (CUDA Toolkit)

Come si accede a CUDA?

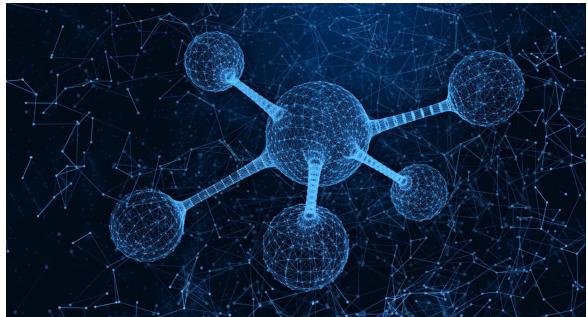
- API (CUDA Runtime API, CUDA Driver API)
- Estensioni a C, C++, Fortran
- Integrazione con framework di alto livello (TensorFlow, PyTorch)

Vantaggio chiave

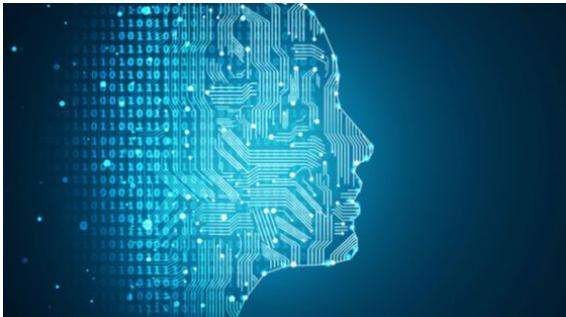
- Accesso alla GPU per calcoli generali, non solo grafica



Settori con Applicazioni Accelerate da CUDA



Computational Chemistry



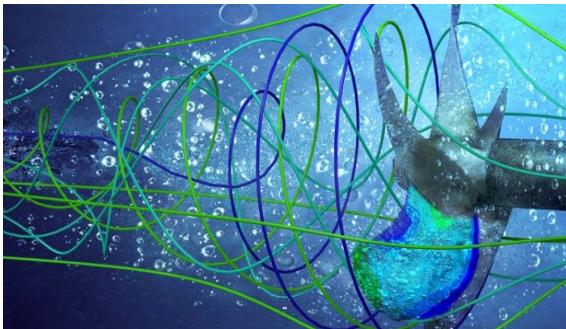
Machine/Deep Learning



Data Science



Bioinformatics



Computational Fluid Dynamics



Computer Vision

Perchè Scegliere la Piattaforma CUDA?

1. Dominio di Mercato e Standard Industriale

- **Standard de facto** per calcolo parallelo e GPU, ampiamente supportata da software e librerie come TensorFlow e PyTorch. Le GPU NVIDIA sono prevalenti in HPC e AI.

2. Prestazioni Elevate

- CUDA consente di sfruttare la potenza delle GPU NVIDIA per eseguire milioni di thread simultaneamente, migliorando significativamente le prestazioni rispetto ai processori CPU tradizionali.

3. Ampia Documentazione e Risorse

- NVIDIA fornisce una **documentazione dettagliata** e risorse pratiche, mentre la comunità attiva consente il supporto e la condivisione di conoscenze tra sviluppatori.

4. Facilità d'Uso

- CUDA estende i linguaggi di programmazione C, C++, e Fortran, permettendo agli sviluppatori di utilizzare **sintassi e concetti già noti**.

5. Versatilità

- È **utilizzato in vari campi**, dalla grafica 3D alla simulazione scientifica, dall'elaborazione video al deep learning, rendendolo una scelta flessibile per molti progetti.

6. Ecosistema Ricco

- CUDA offre un ampio **set di librerie ottimizzate e strumenti di sviluppo** per facilitare l'ottimizzazione e il debugging.

CUDA Toolkit

Cos'è il CUDA Toolkit?

- Il **CUDA Toolkit** è un insieme completo di strumenti di sviluppo fornito da NVIDIA per creare applicazioni accelerate dalla GPU
- È essenziale per lo sviluppo di applicazioni CUDA, poiché fornisce tutti gli strumenti necessari per **scrivere, compilare e ottimizzare** codice CUDA.

Componenti chiave del CUDA Toolkit

Driver NVIDIA

- **Fondamento Invisibile:** Essenziali per CUDA, ma gli sviluppatori raramente interagiscono direttamente con essi.
- **Ruolo:** Funzionano da ponte tra il sistema operativo e la GPU, gestendo l'hardware, il caricamento del codice e il trasferimento dati tra CPU e GPU.
- **Installazione:** Necessari per CUDA e di solito installati separatamente.
- **Compatibilità:** Devono essere compatibili sia con la versione del CUDA Toolkit utilizzata che con la GPU in uso.

CUDA Runtime / CUDA Driver API (Application Programming Interface)

- Gli sviluppatori possono scegliere tra due interfacce per interagire con la GPU:
 - **CUDA Runtime API:** Livello di astrazione più alto, più semplice da utilizzare.
 - **CUDA Driver API:** Livello di astrazione più basso, offre un controllo più granulare sulle operazioni.

CUDA Toolkit

Compilatore CUDA (NVIDIA CUDA Compiler - NVCC)

- **Traduzione del codice:** Compila il codice CUDA, scritto in linguaggi come C, C++ e Fortran, in un formato eseguibile dalle GPU NVIDIA.
- **Fasi:**
 - **Separazione:** Separa il codice destinato alla CPU da quello per la GPU.
 - **Compilazione:** Compila il codice GPU in PTX (linguaggio intermedio) o direttamente in codice macchina per l'architettura GPU target (tenendo conto della **Compute Capability** e della **CUDA Version** utilizzata).
 - **Linking:** Combina il codice CPU e GPU con le librerie CUDA per creare l'applicazione finale.

Librerie CUDA

- **Funzionalità ottimizzate:** Offrono un'ampia gamma di funzioni di alto livello per il calcolo parallelo, ottimizzate per le GPU NVIDIA.
- **Esempi:**
 - **cuBLAS:** Algebra lineare (operazioni su matrici e vettori).
 - **cuFFT:** Fast Fourier Transform (analisi di segnali, elaborazione di immagini).
 - **cuDNN:** Primitive per deep neural networks (convoluzione, pooling, attivazione).
 - **cuRAND:** Generazione di numeri casuali (simulazioni Monte Carlo, crittografia).
 - **cuSPARSE:** Operazioni su matrici sparse (risoluzione di sistemi lineari sparsi).
 - **Thrust:** Algoritmi paralleli generici (ordinamento, ricerca, trasformazioni) su GPU.

CUDA Toolkit

Esempi di Codice (CUDA Samples)

- **Utilità:** Forniscono implementazioni concrete di algoritmi e applicazioni comuni che utilizzano CUDA.
- **Scopo:** Aiutano gli sviluppatori a imparare le best practice di programmazione CUDA e a iniziare rapidamente nuovi progetti.

Strumenti di Debugging e Profiling

- **Nsight Systems:**
 - Analisi a livello di sistema. Offre una visione d'insieme del comportamento dell'applicazione su CPU e GPU, evidenziando eventuali colli di bottiglia.
- **Nsight Compute:**
 - Profiling approfondito della GPU. Permette di analizzare le prestazioni dei kernel CUDA in dettaglio, identificando aree di ottimizzazione per la memoria e l'utilizzo dei core.
- **CUDA-GDB:**
 - Debugging a riga di comando. Permette di eseguire il debug del codice CUDA a livello di sorgente, sia sulla CPU che sulla GPU.
- **NVIDIA Visual Profiler (NVVP) [non più supportato]:**
 - Offre una rappresentazione grafica timeline delle attività della CPU e della GPU, facilitando l'identificazione dei colli di bottiglia.

CUDA Toolkit

Relazione tra CUDA Toolkit e CUDA Version

- Il CUDA Toolkit viene rilasciato in **versioni numerate** (es. CUDA 12.6, CUDA 11.0), determinando la '**CUDA Version**' in uso.
- Ogni nuova CUDA Version include **aggiornamenti software, nuovi strumenti, e supporto** per le ultime architetture GPU.
- Aggiornare il CUDA Toolkit alla versione più recente **garantisce compatibilità con le nuove GPU** e l'accesso alle **ultime ottimizzazioni** per migliorare la performance del codice.

Retrocompatibilità del CUDA Toolkit

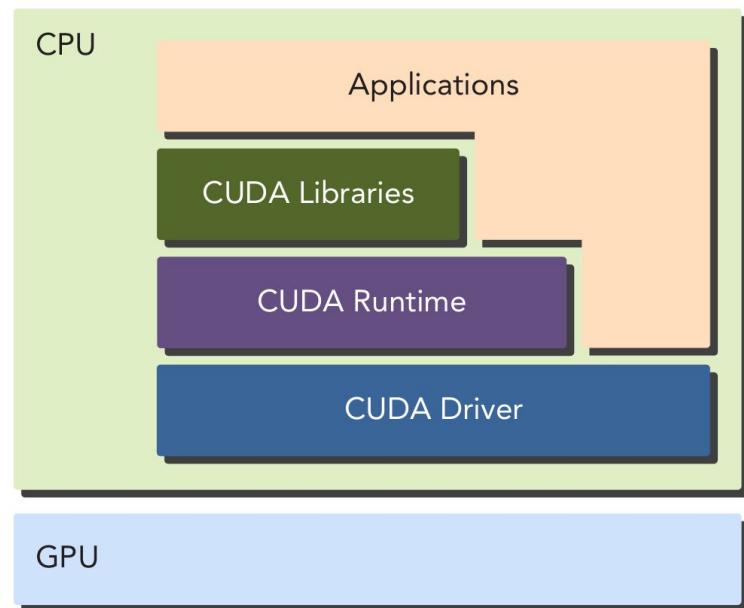
- Supporto per GPU Precedenti:** Le nuove versioni del CUDA Toolkit mantengono la compatibilità con GPU più vecchie, anche se non tutte le funzionalità più recenti sono disponibili su queste architetture.
- Limitazioni:** Alcune funzionalità avanzate introdotte nelle nuove versioni potrebbero non essere supportate su GPU datate, e il **supporto** per architetture molto vecchie può essere **gradualmente ridotto o deprecato**.
- Compatibilità del Codice:** Il codice scritto con versioni precedenti del Toolkit può essere eseguito su versioni più recenti, ma può richiedere piccoli adattamenti.

Introduzione alle API CUDA

- Per sfruttare appieno le capacità delle GPU, è essenziale avere strumenti che facilitino la programmazione.
- Le API forniscono questi strumenti, permettendo ai programmatore di interagire con la GPU in modo efficiente.

Cos'è un'API in CUDA?

- **API** = Application Programming Interface
- Un set di funzioni e procedure che permettono ai programmatore di:
 - **Comunicare** con la GPU
 - **Gestire le risorse** della GPU (memoria, thread, ecc.)
 - **Eseguire** calcoli paralleli sulla GPU

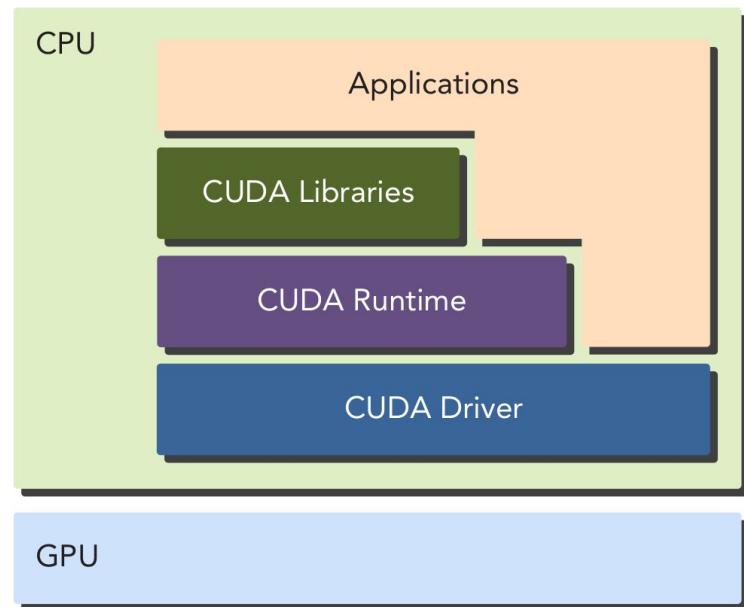


Introduzione alle API CUDA

- Per sfruttare appieno le capacità delle GPU, è essenziale avere strumenti che facilitino la programmazione.
- Le API forniscono questi strumenti, permettendo ai programmatori di interagire con la GPU in modo efficiente.

Due livelli di API in CUDA

- **CUDA Runtime API**
 - Livello più alto e semplice da usare.
 - Gestisce automaticamente molti dettagli di basso livello.
- **CUDA Driver API**
 - Livello più basso e complesso.
 - Offre maggior controllo sulle operazioni della GPU.



Introduzione alle API CUDA

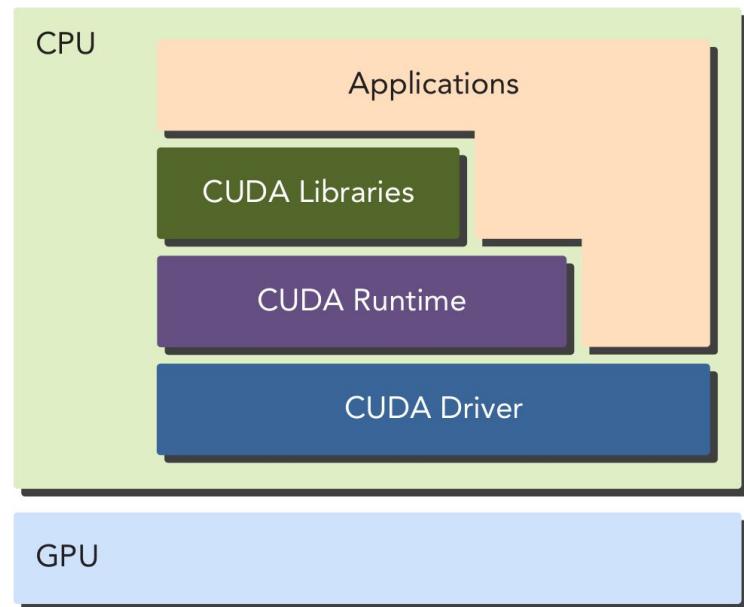
- Per sfruttare appieno le capacità delle GPU, è essenziale avere strumenti che facilitino la programmazione.
- Le API forniscono questi strumenti, permettendo ai programmatori di interagire con la GPU in modo efficiente.

Perché due livelli di API?

- Per bilanciare **semplicità d'uso e controllo dettagliato**.
- Adatto a diverse esigenze e **livelli di esperienza** dei programmatori.

Scelta dell'API

- Dipende dalle esigenze specifiche del progetto

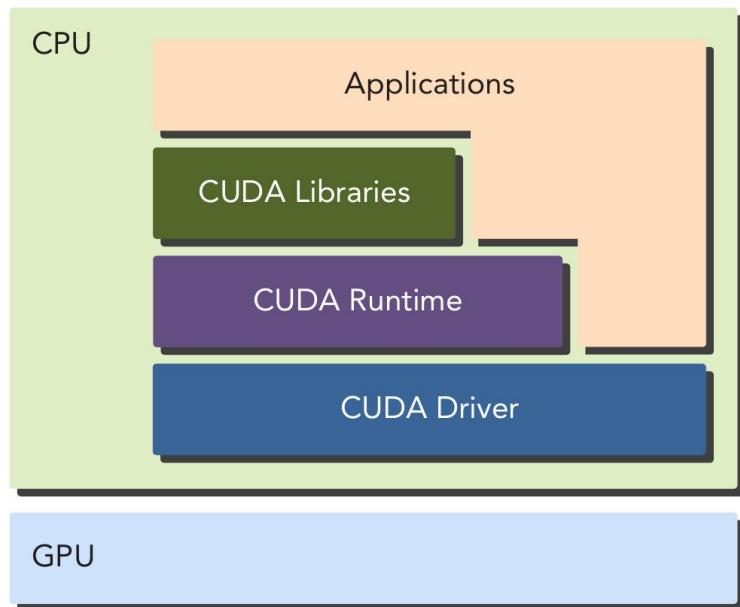


Introduzione alle API CUDA

- Per sfruttare appieno le capacità delle GPU, è essenziale avere strumenti che facilitino la programmazione.
- Le API forniscono questi strumenti, permettendo ai programmatori di interagire con la GPU in modo efficiente.

Relazione tra le due API

- Le funzioni della Runtime API si basano su operazioni di base della Driver API.
- Ogni chiamata della Runtime API è tradotta in più operazioni della Driver API per gestire dettagli complessi.
- La Runtime API è un **wrapper** di alto livello che semplifica l'uso della Driver API, gestendo automaticamente i dettagli complessi.



Introduzione alle API CUDA

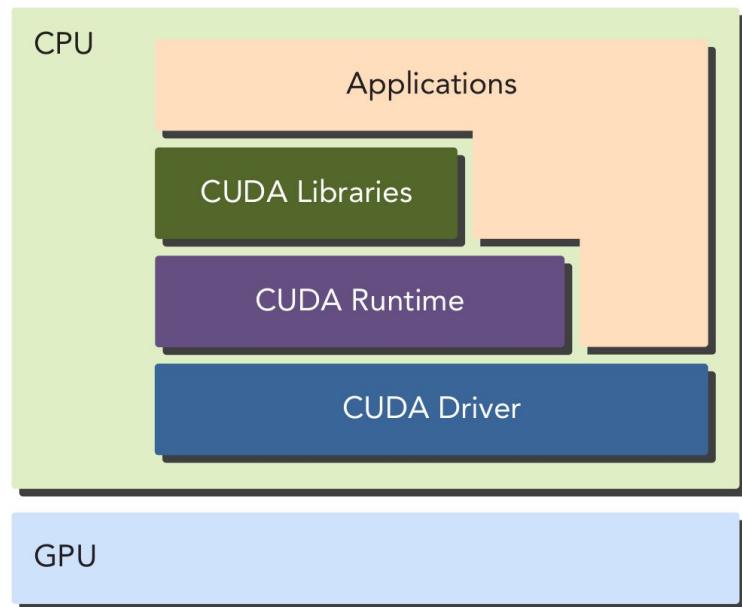
- Per sfruttare appieno le capacità delle GPU, è essenziale avere strumenti che facilitino la programmazione.
- Le API forniscono questi strumenti, permettendo ai programmatori di interagire con la GPU in modo efficiente.

Esclusività

- Le due API sono mutuamente esclusive
- Non è possibile mescolare chiamate di funzione da entrambe le API nello stesso programma

Esempi di semplificazioni offerte dalla Runtime API

- Inizializzazione automatica del dispositivo CUDA
- Gestione implicita del contesto CUDA
- Caricamento automatico dei moduli kernel

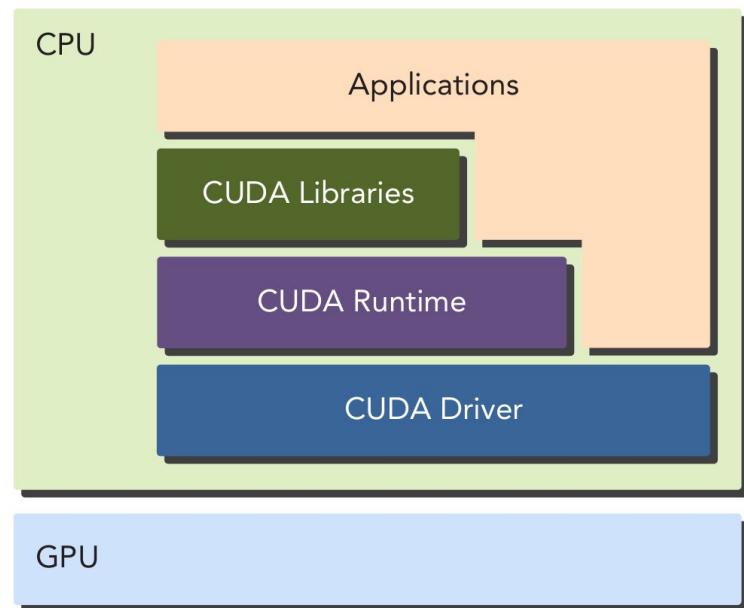


Introduzione alle API CUDA

- Per sfruttare appieno le capacità delle GPU, è essenziale avere strumenti che facilitino la programmazione.
- Le API forniscono questi strumenti, permettendo ai programmatori di interagire con la GPU in modo efficiente.

Prestazioni

- Nessuna differenza significativa di prestazioni tra Runtime e Driver API
- Fattori chiave per le prestazioni:
 - Utilizzo efficiente della memoria nei kernel
 - Organizzazione ottimale dei thread sul dispositivo



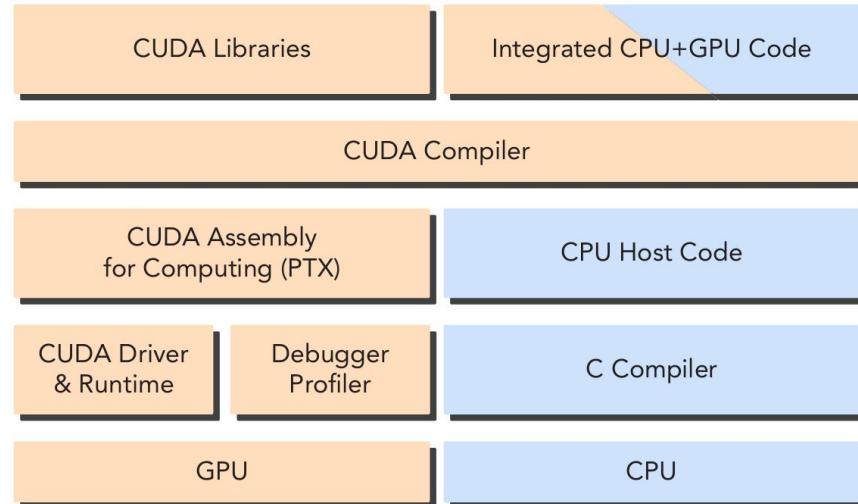
Anatomia di un Programma CUDA

Struttura del Codice Sorgente

- **File Sorgente:** Estensione `.cu`
- Codice host + Codice device

Componenti Principali

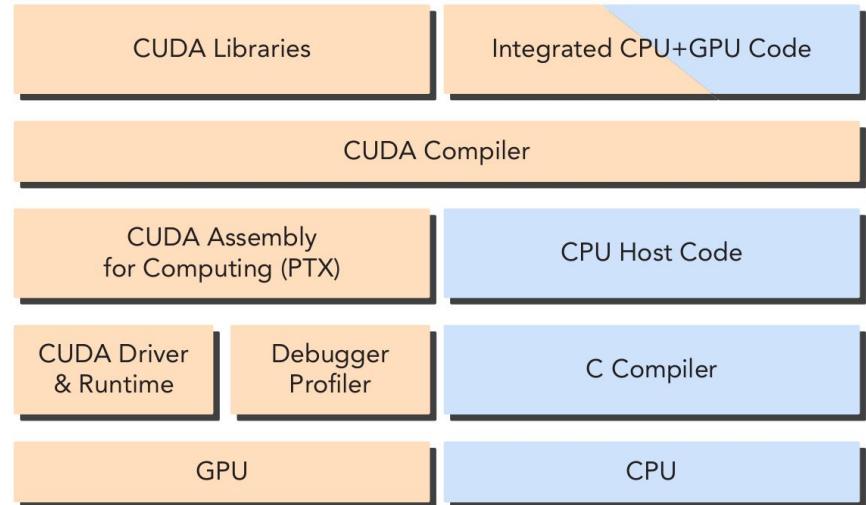
- **Codice Host**
 - Codice C/C++ eseguito sulla CPU.
 - Gestisce la logica dell'applicazione
 - Alloca memoria sulla GPU
 - Trasferisce dati tra CPU e GPU
 - Lancia i kernel GPU
 - Gestisce la sincronizzazione
- **Codice Device:**
 - Codice CUDA C eseguito sulla GPU.
 - Contiene i **kernel** (funzioni parallele)
 - Esegue operazioni computazionali intensive in parallelo



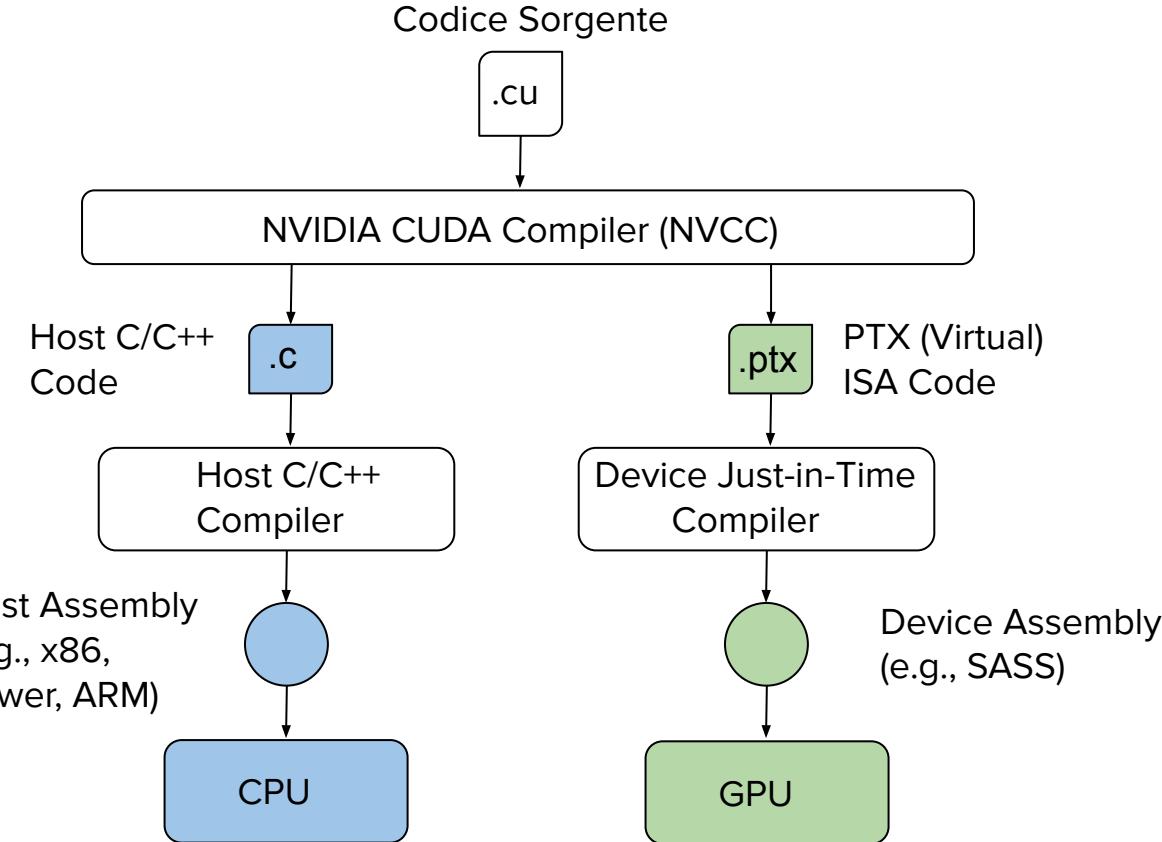
Anatomia di un Programma CUDA

Flusso di Compilazione

- **Separazione del codice**
 - Il compilatore NVIDIA CUDA Compiler (*nvcc*) separa il codice **device** dal codice **host**
- **Compilazione del codice host**
 - È codice C standard o C++
 - Compilato con **compilatori C tradizionali** (gcc)
- **Compilazione del codice device**
 - Compilato da nvcc in formato intermedio **PTX** (Parallel Thread Execution) .
 - Il driver NVIDIA poi traduce il PTX in codice macchina specifico per la GPU (**SASS** - Shader Assembly?) al momento dell'esecuzione, usando un **compilatore Just-In-Time (JIT)**.
- **Fase di linking**
 - Aggiunta delle **librerie runtime CUDA**
 - Supporto per **chiamate ai kernel** e **manipolazione esplicita della GPU**
- **Eseguibile finale**
 - File unico con codice per CPU e GPU



Compilazione di un Programma CUDA



Verifica dell'Ambiente CUDA: Compilatore e Dispositivi GPU

Controllo delle GPU CUDA

- Lista i dispositivi NVIDIA presenti nel sistema
- Nell'esempio: due GPU (nvidia0, nvidia1) e dispositivi di controllo

```
$ ls -l /dev/nv*  
crw-rw-rw- 1 root root 195,    0 Jul 3 13:44 /dev/nvidia0  
crw-rw-rw- 1 root root 195,    1 Jul 3 13:44 /dev/nvidia1  
crw-rw-rw- 1 root root 195, 255 Jul 3 13:44 /dev/nvidiactl  
crw-rw---- 1 root root   10, 144 Jul 3 13:39 /dev/nvram
```

Controllo del Compilatore CUDA

- Verifica la presenza e la posizione del compilatore CUDA (nvcc)

```
$ which nvcc  
/usr/local/cuda/bin/nvc
```

Utilizzo di nvidia-smi per Monitorare le GPU

Gestione e Monitoraggio

- Lo strumento a riga di comando `nvidia-smi` (**NVIDIA System Management Interface**) permette di gestire e monitorare i dispositivi GPU.
- **Comando Base:**

```
$ nvidia-smi -L
```

- Esempio di output:

```
GPU 0: NVIDIA GeForce RTX 3090 (UUID: GPU-276a8f8d-30c8-1bfa-d704-ae8f201e2025)
GPU 1: NVIDIA GeForce RTX 3090 (UUID: GPU-41dc9220-d1b9-69e2-4689-b4ce4a0bb5f2)
```

Query Dettagliate

- Per ottenere dettagli su una GPU specifica:

```
$ nvidia-smi -q -i <ID GPU>
```

- È possibile filtrare le informazioni usando opzioni come **-d MEMORY**, **-d UTILIZATION**, ecc.

```
$ nvidia-smi -q -i 0 -d MEMORY (solo informazioni sulla memoria)
```

```
$ nvidia-smi -q -i 0 -d UTILIZATION (solo informazioni sull'utilizzo)
```

Utilizzo di nvidia-smi per Monitorare Live le GPU

Esempio di Output di nvidia-smi

- Comando Base:

```
$ nvidia-smi
```

NVIDIA-SMI <u>535.183.01</u>			Driver Version: <u>535.183.01</u>		CUDA Version: 12.2		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.
0	NVIDIA GeForce RTX 3090	Off	00000000:17:00.0	Off	0%	N/A	N/A
30%	40C	P8	23W / 350W	10MiB / 24576MiB	Default		N/A
1	NVIDIA GeForce RTX 3090	Off	00000000:65:00.0	Off	0%	N/A	N/A
30%	43C	P8	16W / 350W	76MiB / 24576MiB	Default		N/A
Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage	
ID	ID						
0	N/A	N/A	1234	G	/usr/lib/xorg/Xorg	4MiB	
1	N/A	N/A	1234	G	/usr/lib/xorg/Xorg	56MiB	
1	N/A	N/A	1486	G	/usr/bin/gnome-shell	11MiB	

Utilizzo di nvidia-smi per Monitorare Live le GPU

Monitoraggio Live con `watch`

- Comando:

```
$ watch -n 1 nvidia-smi
```
- **Funzionalità:** Aggiorna l'output di nvidia-smi ogni secondo (modifica l'intervallo con `-n <secondi>`).
- **Utilità:** Monitorare in tempo reale l'utilizzo di GPU, memoria, temperatura, ecc. durante l'esecuzione di un'applicazione CUDA.

Differenze Chiave

- `nvidia-smi`: Fornisce **un'istantanea** dello stato delle GPU al momento dell'esecuzione.
- `watch nvidia-smi`: Offre un **monitoraggio continuo e aggiornato** dello stato delle GPU.

Utilizzo di nvidia-smi per Monitorare le GPU

Configurazione delle GPU a Runtime

- **Sistemi multi-GPU:** CUDA_VISIBLE_DEVICES permette di specificare quali GPU utilizzare a runtime.

Esempi di Configurazione

- **Singola GPU**

- Per usare solo la GPU con ID 2

```
$ export CUDA_VISIBLE_DEVICES=2
```

- La GPU 2 sarà vista dall'applicazione come GPU 0.

- **Multiple GPU**

- Per usare le GPU 2 e 3

```
$ export CUDA_VISIBLE_DEVICES=2,3
```

- Le GPU 2 e 3 saranno mappate come GPU 0 e 1 per l'applicazione.

Vantaggi

- **Flessibilità:** Permette di testare e distribuire il carico su GPU specifiche senza modificare il codice.
- **Isolamento:** Mantiene le altre GPU libere per altre attività o test.

Hello World in CUDA C

Passo 1: Creare il File Sorgente

- Nome file: `hello.cu`

Passo 2: Scrivere il codice

- Codice C e CUDA C a confronto

```
#include <stdio.h>

int main(void) {
    printf("Hello World from CPU!\n");
    return 0;
}
```

Linguaggio C

```
#include <stdio.h>

__global__ void helloFromGPU()
{
    printf("Hello World from GPU thread %d!\n", threadIdx.x);
}

int main()
{
    // Lancio del kernel
    helloFromGPU<<<1, 10>>>();

    // Attendere che la GPU finisca
    cudaDeviceSynchronize();

    return 0;
}
```

Linguaggio CUDA C

Hello World in CUDA C - Analisi

- **Definizione kernel GPU:**
 - **`__global__`**: Qualificatore CUDA per funzioni eseguite sulla GPU e chiamate dalla CPU. Non è presente in C standard.
 - **`threadIdx.x`**: Variabile built-in CUDA che fornisce l'ID univoco del thread all'interno del blocco

```
__global__ void helloFromGPU() {
    printf("Hello World from GPU thread %d!\n", threadIdx.x);
```

- **Funzione main**: Punto di ingresso del programma, eseguito sulla CPU
- **Lancio del kernel**
 - **`<<<1, 10>>`**: Configurazione di esecuzione (1 blocco, 10 thread). Avvia 10 istanze parallele del kernel sulla GPU
- **Sincronizzazione GPU-CPU**
 - **`cudaDeviceSynchronize()`**: Attende il completamento di tutte le operazioni GPU

```
int main()
{
    helloFromGPU<<<1, 10>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

Hello World in CUDA C

Passo 3: Compilazione

- Salvare il codice nel file `hello.cu`
 - Un file .cu può contenere sia codice C/C++ standard che codice CUDA C
 - Questo permette di **mescolare codice** per CPU e GPU nello stesso file
- Compilare il programma usando il compilatore CUDA nvcc
 - nvcc può gestire sia il codice C/C++ standard che le estensioni CUDA
 - nvcc separa internamente il codice host e device, compilando ciascuno in modo appropriato

```
$ nvcc hello.cu -o hello
```

Hello World in CUDA C

Passo 4: Esecuzione

- Eseguire il file eseguibile:

```
$ ./hello
```

• Passo 4: Esecuzione

- Output:

Hello World from CPU!

Variante Linguaggio C

Hello World from GPU thread 0!
Hello World from GPU thread 1!
Hello World from GPU thread 2!
Hello World from GPU thread 3!
Hello World from GPU thread 4!
Hello World from GPU thread 5!
Hello World from GPU thread 6!
Hello World from GPU thread 7!
Hello World from GPU thread 8!
Hello World from GPU thread 9!

Variante Linguaggio CUDA C

Ottenere Informazioni sulla GPU tramite API CUDA

Utilizzo delle API CUDA

- CUDA fornisce API per ottenere **informazioni dettagliate** sulle GPU direttamente dal codice

```
int main() {
    cudaDeviceProp prop;
    cudaGetDeviceProperties(&prop, 0); // Ottieni proprietà del dispositivo 0

    printf("Nome Dispositivo: %s\n", prop.name);
    printf("Memoria Gloable Totale: %.0f MB\n", prop.totalGlobalMem / 1024.0 / 1024.0);
    printf("Clock Core: %d MHz\n", prop.clockRate / 1000);
    printf("Compute Capability: %d.%d\n", prop.major, prop.minor);
    // Stampa di altre proprietà
    return 0;
}
```

- Utilizzo della struttura **cudaDeviceProp** per memorizzare le proprietà del dispositivo
- Utilizzo della funzione **cudaGetDeviceProperties** per ottenere le proprietà del dispositivo specificato
- Accesso alle proprietà della GPU, come **nome**, **memoria totale**, **clock core** e **compute capability**
- Per una lista completa delle proprietà disponibili, consulta la **documentazione ufficiale di CUDA**:
 - <https://docs.nvidia.com/cuda/cuda-runtime-api/structcudaDeviceProp.html>

Ottenere Informazioni sulla GPU tramite API CUDA

Esempio di Output

```
CUDA System Information:  
CUDA Driver Version: 12.2  
CUDA Runtime Version: 11.3  
Numero di dispositivi CUDA: 2  
  
Dispositivo 0: NVIDIA GeForce RTX 3090  
1. Compute Capability: 8.6  
2. Memoria Globale Totale: 23.69 GB  
3. Numero di Multiprocessori: 82  
4. Clock Core: 1695 MHz  
5. Clock Memoria: 9751 MHz  
6. Larghezza Bus Memoria: 384 bit  
7. Dimensione Cache L2: 6144 KB  
8. Memoria Condivisa per Blocco: 48 KB  
9. Numero Massimo di Thread per Blocco: 1024  
10. Dimensioni Massime Griglia: (2147483647, 65535, 65535)  
11. Dimensioni Massime Blocco: (1024, 1024, 64)  
12. Warp Size: 32  
13. Memoria Costante Totale: 65536 bytes  
14. Texture Alignment: 512 bytes
```

Cosa Significa Programmare in CUDA C?

Pensare in Parallello

- **Decomposizione del Problema:** Identificare le parti del problema che possono essere eseguite in parallelo per sfruttare al meglio le risorse della GPU.
- **Architettura della GPU:** Le GPU sono composte da migliaia di core in grado di eseguire thread in parallelo. CUDA fornisce gli strumenti per organizzare e gestire questi thread.
- **Scalabilità:** Progettare algoritmi che si adattano a diversi numeri di thread (e GPU).
- **Gerarchia di Thread:** Organizzare il lavoro in blocchi e griglie per massimizzare l'efficienza.
- **Gerarchia di Memoria:** Utilizzare strategicamente memoria globale, condivisa e locale per ridurre i tempi di accesso.
- **Sincronizzazione:** Gestire la coordinazione tra thread e il trasferimento dati tra CPU e GPU senza conflitti.
- **Bilanciamento del Carico:** Distribuire il lavoro in modo uniforme per evitare colli di bottiglia.

Scrittura di codice in CUDA C

- Si scrive **codice sequenziale in C**
- **Si estende a migliaia di thread**, permettendo di pensare in termini sequenziali mentre si sfrutta il calcolo parallelo della GPU.

Panoramica del Modulo 2

- **Modello di Programmazione CUDA**
 - Scrivere codice parallelo per GPU con CUDA C/C++, organizzandolo in thread, blocchi e griglie.
- **Modello di Esecuzione CUDA**
 - Come la GPU gestisce ed esegue il codice CUDA
- **Organizzazione delle Memorie in CUDA**
 - Tipi di memoria disponibili in CUDA e il loro utilizzo efficiente.
- **Stream e Concorrenza in CUDA**
 - Utilizzare gli stream per sovrapporre calcolo e trasferimento dati.
- **Librerie e Applicazioni in CUDA**
 - Librerie CUDA ottimizzate per diversi domini e come usarle per problemi reali.

Riferimenti Bibliografici

Testi Generali

- Cheng, J., Grossman, M., McKercher, T. (2014). **Professional CUDA C Programming**. Wrox Pr Inc. (1^a edizione)
- Kirk, D. B., Hwu, W. W. (2013). **Programming Massively Parallel Processors**. Morgan Kaufmann (3^a edizione)

NVIDIA Docs

- CUDA Programming:
<http://docs.nvidia.com/cuda/cuda-c-programming-guide/>
- CUDA C Best Practices Guide
<http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/>

Risorse Online

- Corso GPU Computing (Prof. G. Grossi): Dipartimento di Informatica, Università degli Studi di Milano
 - <http://gpu.di.unimi.it/lezioni.html>