UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA

**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE**

**CORSO DI LAUREA IN INGEGNERIA INFORMATICA**

# "UTILIZZO DI PROCESSORI GRAFICI PER SISTEMI DI INTELLIGENZA ARTIFICIALE"

**Relatore: Prof. / Dott. Zanoni Enrico**

**Laureando/a: Lambertini Lorenzo**

**ANNO ACCADEMICO 2022 - 2023**

**Data di laurea 13 marzo 2023**

# Capitolo 1. INDICE

# Introduzione

Chiunque è circondato dai molteplici utilizzi dell'intelligenza artificiale, basti pensare agli assistenti vocali, ai nuovi sistemi di sicurezza di alcune auto elettriche o ai riconoscimenti facciali dei nostri smartphone. Molti però non sanno che queste non sono che le ultime applicazioni di questa tecnologia. Le aziende spesso utilizzano queste IA in molti prodotti senza che l'utente finale se ne accorga, come alcuni servizi di streaming che riescono a consigliarci loro prodotti in base alle preferenze che abbiamo dimostrato precedentemente o alcuni smartphone che utilizzano IA per migliorare le foto fatte dall'utente.

L'utilizzo dei processori grafici (GPU, chiamati anche schede video) nell'ambito dell'intelligenza artificiale è un argomento di grande importanza, poiché queste sono diventate uno strumento fondamentale per il training di modelli di intelligenza artificiale complessi. L'obiettivo di questa tesi è capire il funzionamento alla base delle IA, analizzare la struttura e il modo di operare delle GPU, capire perché sono un'opzione migliore delle CPU e quindi analizzare il loro l'impatto sulle prestazioni dei modelli di intelligenza artificiale.

# Capitolo 1 Intelligenza artificiale

## 1.1 Il settore informatico

Il settore informatico dell'intelligenza artificiale si occupa della progettazione e della programmazione di sistemi hardware e software con lo scopo di dotare le macchine di caratteristiche tipicamente umane come, ad esempio, la capacità di prendere decisioni. La data di nascita di questo settore viene fissata nel 1956, quando durante un convegno negli Stati Uniti si utilizzò per la prima volta il termine "Sistema Intelligente", in particolare con la presentazione del programma Logic Theorist dei due ricercatori informatici Allen Newell e Hebert Simon, il quale era in grado di dimostrare semplici teoremi di matematica partendo da determinate informazioni fornite in input. Nei successivi anni questo settore si espanse tanto da interessare aziende come IBM. Nacquero così programmi sempre più sofisticati e Lisp, il primo linguaggio di programmazione pensato per l'intelligenza artificiale.

```
CL-USER> (defun hello ()
           (format t "Hello, World!~%"))
HELLO
CL-USER> (hello)
Hello, World!
NIL
CL-USER>
```

Figura 1 Stampa in output di "Hello, World!" in Lisp

## 1.2 Forte e debole

Il concetto di IA si ramifica in due teorie distinte chiamate intelligenza artificiale forte e intelligenza artificiale debole. La prima si basa sull'ipotesi secondo cui le macchine saranno in grado di avere coscienza di sé e di replicare il pensare umano, mentre la seconda ritiene possibile sviluppare macchine in grado di risolvere problemi senza la coscienza delle attività svolte, ovvero sistemi in grado di svolgere funzione umane complesse ma non di pensare. Per quanto esistano diversi software che sono in grado di simulare una IA forte questa deve ancora venire alla luce, al contrario della sua controparte debole che viene largamente utilizzata per una moltitudine di applicazioni.

## 1.3 Reti neurali

Un'importante svolta nel mondo delle IA si ha con l'avvento delle reti neurali, le quali hanno l'idea di simulare il funzionamento dei neuroni all'interno del cervello umano, permettendo prestazioni impossibili con altri algoritmi.

Questa innovazione (chiamate anche ANN, ovvero Artificial Neural Network) si basa su una struttura a livelli composti da nodi (che simulano i corpi cellulari dei neuroni) e dai collegamenti fra livelli (che simulano gli assoni dei neuroni). I livelli si dividono in input layer, hidden layers e output layer, mentre le connessioni variano in base alla tipologia di rete neurale utilizzata.

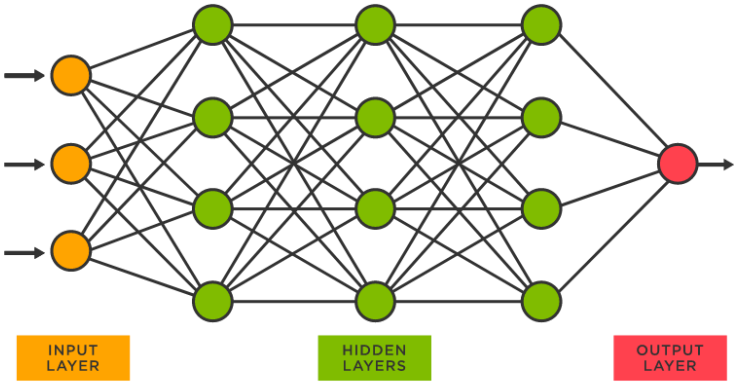


Figura 2 Esempio di rete neurale

Proprio come il sistema nervoso le ANN funzionano tramite due meccaniche principali, ovvero il sistema di peso dei collegamenti fra neuroni di livello diverso e la funzione di soglia dei neuroni stessi: forniti i dati nel livello di input i neuroni manderanno o meno un segnale in base alla loro soglia, ovvero un valore minimo necessario all'attivazione di del neurone stesso e se ciò avviene il segnale verrà moltiplicato per il peso del collegamento fra l'input e il primo hidden layer; similmente ai neuroni di input, quelli degli strati intermedi manderanno il segnale in base al loro valore di soglia, ma quest'ultimo non deriva da dati forniti in input, ma dai segnali dello strato precedente. Per calcolare il segnale che ogni neurone manderà si utilizza la funzione di soglia, la quale solitamente somma i segnali di ingresso con un bias (un valore di default deciso precedentemente dal programmatore e modificato successivamente):

$$\text{output} = f(x) = \begin{cases} 1 \text{ se } \sum w_i x_i + b \geq 0 \\ 0 \text{ se } \sum w_i x_i + b < 0 \end{cases}$$

Nell'ultimo livello della rete neurale avremo ogni neurone con un valore diverso e in base a questo avremo il risultato finale (esempio: in caso di riconoscimento di un oggetto il nodo finale con il valore più alto corrisponderà all'oggetto che la rete ha riconosciuto).

## 1.4 Apprendimento automatico

Una rete neurale migliora le sue prestazioni modificando parametri come i pesi delle connessioni e le funzioni di soglia, e per fare questo si dice che la rete deve essere allenata. Esistono molteplici tipologie di allenamento di una rete, chiamati apprendimenti, tra le quali supervisionato, non supervisionato e di rinforzo.

## 1.5 Apprendimento supervisionato

L'apprendimento supervisionato sussiste nel fornire al sistema un insieme di dati specifici e codificati (training set) i quali permettono alla macchina di addestrarsi tramite un algoritmo, solitamente di back propagation, modificando i propri parametri allo scopo di minimizzare l'errore relativo al set di dati di allenamento. L'obiettivo finale di questa tipologia di apprendimento è la previsione del valore d'output dato un certo input valido, come ad esempio problemi di classificazione o regressione.

## 1.6 Apprendimento non supervisionato

L'apprendimento non supervisionato si basa su algoritmi che modificano i parametri della rete tenendo conto dei dati contenenti solamente variabili di ingresso. In questi casi gli algoritmi cercano di suddividere i dati in cluster, ovvero categorie che suddividono e rappresentano ciò che viene passato in input.

## 1.7 Apprendimento con rinforzo

L'apprendimento con rinforzo rappresenta un sistema d'allenamento più complesso che prevede di migliorare le prestazioni della macchina prendendo in considerazione un feedback generato dall'ambiente esterno al sistema. Questo può essere positivo o negativo, ovvero può incentivare o meno un'eventuale evoluzione del sistema.

## 1.8 Apprendimento approfondito

L’apprendimento approfondito, o deep learning, viene utilizzato su reti neurali particolarmente grandi (molti hidden layers) e si basa sull’apprendimento di dati non forniti dall’uomo ma appresi tramite l’utilizzo di altri strumenti, come il calcolo statistico. In questo modo si genera un apprendimento piramidale dove concetti più alti vengono appresi a partire da altri più bassi.

Per permettere un addestramento di livello sufficientemente elevato questa tipologia di apprendimento si necessita una quantità estremamente elevata di dati, i quali necessitano a loro volta di essere analizzati. questo è permesso solo grazie a potenze di calcolo non indifferenti.

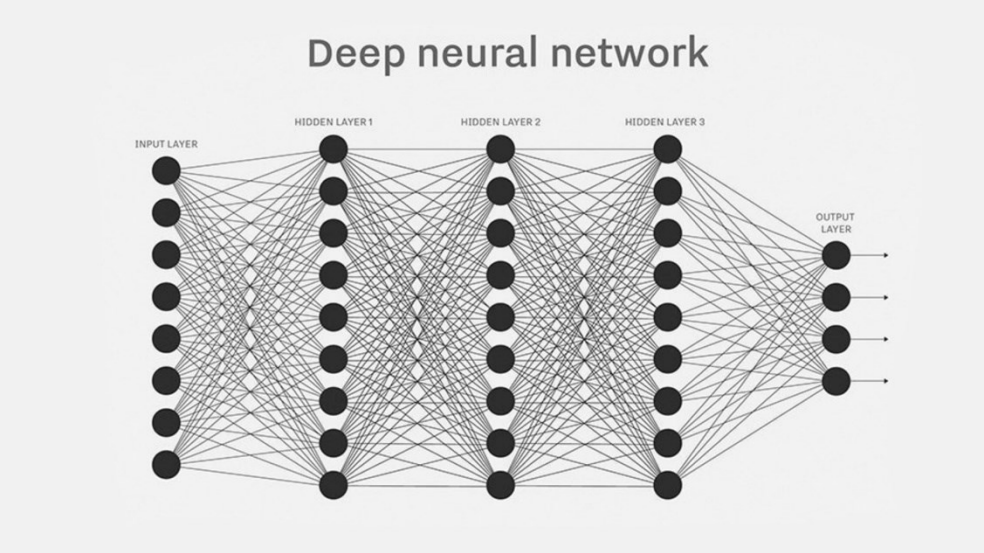


Figura 3 Rete neurale per deep learning

# Capitolo 2 Necessità di potenza di calcolo

## 2.1 I limiti delle CPU

Come abbiamo visto precedentemente i vantaggi offerti dall'intelligenza artificiale, dal machine learning e dal deep learning sono numerosi, ma le prestazioni di questi dipendono dalla mole di dati analizzati e quindi dalla potenza dui calcolo dell'hardware utilizzato.

Per molti anni la CPU (Central Processing Unit) è stata la componente con maggior potenza di calcolo nel mondo dei server e dei PC, e nel corso del tempo le sue prestazioni sono cresciute, ma non abbastanza per ricoprire il ruolo di tuttofare all'interno dei sistemi. I processori centrali infatti nascono per svolgere una vasta gamma di attività, spesso differenti fra loro, ma questo va a discapito della pura potenza di calcolo.

In particolare, basti pensare a come negli anni il reparto di gestione video abbia visto un'evoluzione quasi esponenziale, ad esempio negli ultimi decenni si è passato da gestire video in bassa risoluzione e basso frame rate a video in altissima risoluzione (come il 4k UHD) e alto frame rate (anche 120 fps).

## 2.2 Soluzione: processori grafici

In soccorso delle CPU nacquero le GPU (Graphics Processing Unit), ovvero i processori grafici che hanno lo scopo di accelerare la creazione di immagini snellendo il lavoro gravoso dei processori centrali. Nonostante le GPU nascano con una potenza di calcolo notevolmente maggiore rispetto alle CPU non si possono considerare un rimpiazzo, anzi non sono altro che un aiutante. Infatti, a differenza dei processori centrali, le schede video vengono progettato con lo scopo di eseguire una gamma minore di attività, ma molto più velocemente utilizzando la parallelizzazione.


Figura 4 Esempio CPU


Figura 5 Esempio GPU

Nel capitolo successivo analizzeremo la struttura e il funzionamento delle GPU, ma prima confronteremo processore centrale e processore grafico negli ambiti visti precedentemente.

## 2.3 CPU vs. GPU per reti neurali

Come già detto le reti neurali imparano da enormi quantità di dati nel tentativo di simulare il comportamento del cervello umano.

Poiché le reti neurali funzionano principalmente con enormi set di dati, il tempo di addestramento può aumentare man mano che il set di dati cresce. Sebbene sia possibile, ma sconsigliato, addestrare reti neurali su scala ridotta utilizzando le CPU, le CPU diventano meno efficienti nell'elaborazione di questi grandi volumi di dati, causando un aumento del tempo di addestramento quando vengono aggiunti più livelli e parametri.

Le reti neurali costituiscono la base del deep learning (una rete neurale con tre o più livelli) e sono progettate per essere eseguite in parallelo, con ciascuna attività in esecuzione indipendentemente dall'altra. Ciò rende le GPU più adatte all'elaborazione degli enormi set di dati e dei complessi dati matematici utilizzati per addestrare le reti neurali.

## 2.4 CPU vs. GPU per machine learning

Abbiamo visto che le CPU non sono efficienti per i processi di apprendimento automatico ad alta intensità di dati, ma sono comunque un'opzione conveniente quando l'utilizzo di una GPU non è ideale.

Tali casi includono algoritmi di apprendimento automatico, come dati di serie temporali, che non richiedono il calcolo parallelo, nonché sistemi di raccomandazione per l'addestramento che richiedono molta memoria per l'incorporamento di livelli. Alcuni algoritmi sono anche ottimizzati per utilizzare le CPU rispetto alle GPU.

Più dati si processano, meglio e più velocemente può apprendere un algoritmo di machine learning. La tecnologia delle GPU è andata oltre l'elaborazione di grafica ad alte prestazioni per utilizzare casi che richiedono elaborazione dati ad alta velocità e calcoli massicciamente paralleli. Di conseguenza, le GPU forniscono l'elaborazione parallela necessaria per supportare i complessi processi multifase coinvolti nell'apprendimento automatico.

## 2.5 CPU vs. GPU per deep learning

Come visto precedentemente l'addestramento di reti di deep learning con set di dati di grandi dimensioni può aumentare l’accuratezza della loro previsione.

Le CPU sono meno efficienti delle GPU per il deep learning perché elaborano le attività in ordine una alla volta. Man mano che vengono utilizzati più punti dati per l'input e le previsioni, diventa più difficile per una CPU gestire tutte le attività associate (struttura piramidale dell’apprendimento).

Il deep learning richiede una grande velocità e prestazioni elevate e i modelli apprendono più rapidamente quando tutte le operazioni vengono elaborate contemporaneamente. Poiché hanno migliaia di core, le GPU sono ottimizzate per l'addestramento di modelli di deep learning e possono elaborare più attività parallele fino a tre volte più velocemente di una CPU.

# Capitolo 3 Le schede video

In questo capitolo si analizzano le componenti chiave di una generica GPU per poi analizzare come una moderna scheda video possa lavorare e processare in maniera parallela un'enorme quantità di dati.

## 3.1 Il termine GPU

L'unità di elaborazione grafica (in inglese Graphics Processing Unit, sigla GPU) è un circuito elettronico progettato per accelerare la creazione di immagini in un frame buffer, destinato all'output su un dispositivo di visualizzazione. Le GPU vengono utilizzate in sistemi embedded come telefoni cellulari, personal computer e console di gioco. In un personal computer una GPU può essere presente su scheda video o incorporata sulla scheda madre, mentre in alcune CPU sono incorporate nel die della CPU stessa.

Negli anni Settanta il termine GPU descriveva un'unità di elaborazione programmabile, che lavorava indipendentemente dalla CPU ed era responsabile della manipolazione e dell'output della grafica. Successivamente, nel 1994, Sony ha utilizzato il termine in riferimento alla GPU progettata da Toshiba per la console PlayStation. Il termine GPU è stato reso popolare da NVIDIA nel 1999, che ha commercializzato la GeForce 256 come "la prima GPU al mondo".

## 3.2 Processore grafico

Questo è il cuore della GPU e si occupa di trasformare il segnale elettrico ricevuto dalla CPU in un segnale leggibile ai monitor e televisori, quindi un segnale video digitale.

In certi casi, nei sistemi più economici o in quelli che ne sono sprovvisti, il processore probabilmente disporrà di una GPU integrata che possa permettere di usufruire delle più semplici applicazioni senza l'installazione di una scheda aggiuntiva.

La nascita di queste tecnologie è alla base anche di sistemi come gli smartphone, i quali hanno un unico calcolatore che funge sia da CPU che da GPU. Tali sistemi sono denominati "SoC" (System on a chip). Aziende come Intel o AMD hanno creato processori dotati di grafica integrata mentre diversamente la pensa AMD che preferisce puntare sulle APU (Accelerated Processing Unit), le quali se affiancate ad una scheda Radeon lavorano in sinergia per offrire prestazioni più elevate.

La struttura e il funzionamento di questo componente verranno analizzati in un paragrafo dedicato (vedi paragrafo 3.5).

## 3.3 ROM e Bios

Su ogni scheda video si trova sempre una memoria ROM (Read only memory), nella quale è installato il bios che permetterà alla scheda di avviarsi correttamente ogni qual volta accendiate il computer. Questa fase prende il nome di “boot” o “bootstrap” e permetterà la lettura delle informazioni necessarie per inviare in output i primi segnali video.

## 3.4 RAM

Esattamente come la memoria principale del sistema (sempre denominata RAM) anche in questo caso essa serve da tramite con il fine di velocizzare il sistema e di evita di imbattersi in fastidiosi bottleneck (colli di bottiglia: si verificano quando vi è una differenza prestazionale rilevante tra due componenti, o quando un primo rallenta il secondo) dovuti all’utilizzo da parte di CPU e GPU di un’unica memoria.

Tuttavia, dato che le memorie comuni sono troppo lente per una scheda video, all’interno di esse vengono utilizzate le VRAM (la quale sta andando in disuso), le WRAM (Windows RAM), e le SGRAM (Synchronous Graphic RAM, memoria ad accesso casuale sincronizzata dal prezzo abbastanza contenuto).

Le RAM più utilizzate nel campo delle schede video al momento sono le GDDR5 (comparse per la prima volta con la serie ATI HD4000) le quali utilizzano un sistema di prefetch (ovvero precaricare la prossima istruzione del programma durante l'esecuzione dell'istruzione corrente) e sono studiate appositamente per garantire una elevata velocità di trasferimento. Le più nuove sono le GDDR6, le quali fanno concorrenza ad un’ultima innovazione portata avanti da AMD: ovvero le memorie HBM (High Bandwidth Memory) delle quali il progetto ebbe inizio nel 2008.

Le schede video odierne possiedono in media tra i 2 agli 8Gb di memoria mentre esistono delle proposte molto costose che possono raggiungere dai 24 agli 80 GB, tuttavia la vera differenza si nota solo a definizioni alte in ambiti professionali.

## 3.5 Parallelizzazione

Le GPU sono caratterizzate da una struttura altamente parallela, che consente di manipolare grandi insiemi di dati in maniera efficiente. Questo, in combinazione con la crescita prestazionale dell'hardware grafico e i suoi recenti miglioramenti in termini di programmabilità, ha portato l'attenzione del mondo scientifico sulla possibilità di utilizzare la GPU per scopi diversi da quelli tradizionali. Il settore che si occupa di questo argomento è chiamato GP-GPU (General-Purpose computing on Graphics Processing Unit) e sarà analizzato in dettaglio nel capitolo successivo. Per comprendere meglio la sezione seguente, verrà ora analizzata l'architettura di un generico processore grafico moderno, rappresentata in figura 6.

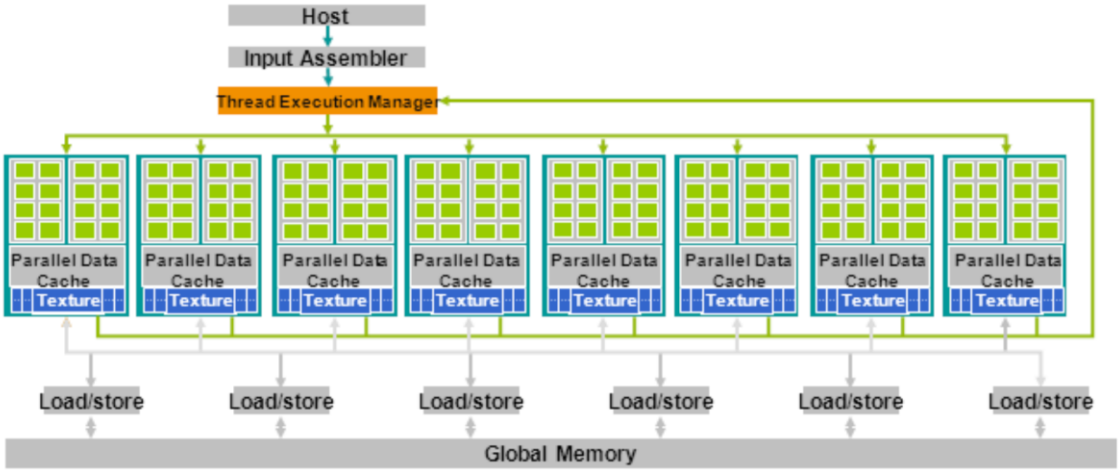


Figura 6 Architettura di una GPU

Le GPU sono coprocessori e in quanto tali devono essere connesse ad un processore centrale (la classica CPU) chiamato, in questo contesto, host. Generalmente il collegamento tra i due dispositivi è realizzato con un bus PCI Express ad alta velocità (uscita da poco la nuova tecnologia PCI Express gen.5 che promette una banda da 128GB/s), che consente il trasferimento di dati dalla memoria della CPU a quella della GPU e viceversa. Come già detto i processori grafici lavorano ad un alto grado di parallelismo, e questo è dovuto in particolare nella struttura dei diversi dispositivi di memorizzazione e nella disposizione dei core d'esecuzione. Ogni GPU è infatti composta da unità di elaborazione chiamate Streaming Multiprocessor (SM), le quali rappresentano il primo livello logico di parallelismo: ogni SM lavora contemporaneamente e in maniera indipendente dagli altri. La SM è a sua volta suddiviso in un gruppo di Streaming

Processor (SP), ognuno dei quali è un core d'esecuzione vero e proprio in grado di eseguire sequenzialmente un thread. La divisione in SM e SP è di natura strutturale, ma è possibile delineare una ulteriore organizzazione logica tra gli SP di una GPU, che sono infatti raggruppati in blocchi logici caratterizzati da una particolare modalità d'esecuzione: tutti i core che compongono un gruppo eseguono contemporaneamente la stessa istruzione. Ogni processore grafico ha inoltre a disposizione diversi tipi di memoria, ognuna delle quali differenziata per posizione, caratteristiche e scopo.

Come abbiamo già visto l'unità di memoria principale è la RAM, la quale promette alta capacità a discapito della latenza. Tutti i core della GPU possono accedere a questo spazio di memoria, così come può fare anche l'host, che vi è collegato direttamente. Per questo motivo la memoria globale è spesso utilizzata per ospitare grandi quantità di dati trasferiti dalla memoria del processore. Non essendo particolarmente veloce, questa memoria spesso dispone di meccanismi di ottimizzazione degli accessi o di livelli di cache, i quali consentono comunicazioni più efficienti con la GPU.

Un altro spazio di memoria accessibile a tutti i core è la texture memory, una regione utilizzabile dalla GPU in sola lettura, originariamente progettata per la memorizzazione di particolari immagini, chiamate texture, impiegate nella rappresentazione grafica di elementi tridimensionali. Questo tipo di memoria dispone di specifici meccanismi di caching che la rendono efficiente nella memorizzazione di strutture bidimensionali (matrici o immagini) e nell'accesso ad elementi spazialmente vicini. All'interno degli SM è presente inoltre uno spazio di memoria condiviso tra i core che compongono un gruppo di lavoro. Questa memoria ha dimensioni molto più ridotte, dell'ordine di qualche decina di MB per ogni SM, ma è decisamente più veloce di quella globale ma e per questo può essere utilizzata per memorizzare valori che devono essere utilizzati più volte, da core diversi, limitando il numero di accessi alla memoria globale. Gli aspetti negativi della condivisione di questo spazio di memoria riguardano i noti problemi che affliggono i modelli di calcolo concorrenti: è necessario assicurare in ogni momento la coerenza dei valori memorizzati, anche a costo di sequenzializzare accessi contemporanei alla stessa locazione di memoria. Ogni SM inoltre dispone di un certo numero di registri, che rappresentano uno spazio di memoria ad accesso rapido, temporaneo, locale (non condiviso tra i core) e di dimensioni limitate, che consente la memorizzazione di valori frequentemente utilizzati da un singolo core.

# Capitolo 4 General Purpose Computing on GPU

Il GP-GPU (Generale Purpose computing on Graphics Processing Unit) è il settore dedicato allo studio delle tecniche che consentono di sfruttare la potenza computazionale delle GPU per effettuare calcoli usufruendo dell'alto livello di parallelismo interno che le contraddistingue. Come visto nel capitolo precedente, le GPU sono strutturate in maniera completamente diversa dai tradizionali processori e per questo presentano problemi di natura differente, richiedono tecniche di programmazione specifiche. La caratteristica più rilevante che contraddistingue i processori grafici è l'alto numero di core a disposizione, che permette di portare avanti molti thread d'esecuzione concorrenti, parzialmente sincronizzati nell'esecuzione della stessa operazione. Questa caratteristica risulta molto utile ed efficiente nelle situazioni in cui è possibile suddividere il lavoro in tante parti, effettuando le stesse operazioni su dati diversi; al contrario, è difficile utilizzare al meglio un'architettura di questo tipo quando esiste una forte sequenzialità e un ordine logico da rispettare nelle operazioni da svolgere, o il lavoro non può essere diviso equamente in tante piccole sottoparti. Il paradigma di programmazione che caratterizza il calcolo su GPU è chiamato Stream Processing, perché i dati possono essere visti come un flusso omogeneo di valori ai quali vengono applicate in maniera sincrona le stesse operazioni. Le funzioni che processano i dati nello stream e che sono eseguite sulla GPU prendono il nome di kernel e sono in grado di applicare solo un insieme limitato di operazioni ad ogni flusso di dati.

Nel modello di programmazione stream processing, un programma principale è in esecuzione sull'host (CPU) e gestisce il parallelismo concatenando esecuzioni di kernel su stream diversi. La struttura di funzionamento di un kernel segue quasi sempre uno schema preciso: per prima cosa viene caricato nella memoria della GPU un flusso di dati (input), sul quale il kernel applica una serie di operazioni che producono un flusso di dati in uscita (output). Sarà poi compito dell'host estrarre dalla memoria della GPU i dati prodotti dal kernel. È evidente quindi che non tutte le applicazioni possono sfruttare a pieno la potenza di calcolo delle GPU perché, per ottenere un reale guadagno computazionale, deve essere possibile una completa parallelizzazione delle operazioni. L'elaborazione GP-GPU raggiunge prestazioni ottimali nella soluzione di problemi di grandi dimensioni, che possono però essere suddivisi in tanti sotto-problemi dello stesso tipo, risolubili contemporaneamente.

## 4.1 Applicazioni GP-GPU

Il grande aumento di performance presentato dalle GPU nell'ultimo decennio ha attirato l'interesse del mondo scientifico e degli sviluppatori verso la possibilità di impiegare i processori grafici come piattaforme per effettuare calcoli general purpose. Questo ha consentito la nascita e il rapido sviluppo di diversi linguaggi di programmazione per l'ambiente grafico. Il GPU computing ha preso veramente il via quando, nel 2006, vennero presentati CUDA e Stream, due interfacce per la programmazione grafica progettate dai due principali sviluppatori di GPU, rispettivamente NVIDIA e AMD (AMD acquisisce ATI nel 2006 e così facendo entra nel settore GPU). Qualche anno più tardi venne avviato il progetto OpenCL, con lo scopo di realizzare un framework d'esecuzione eterogeneo sul quale potessero lavorare sia GPU (sviluppate da diversi produttori), sia CPU. Col passare del tempo questi linguaggi si sono evoluti enormemente, diventando sempre più simili ai comuni linguaggi di programmazione general purpose e aggiungendo meccanismi di controllo sempre più validi ed efficaci. Attualmente, CUDA e OpenCL rappresentano le soluzioni più efficienti per sfruttare in maniera diretta la potenza di calcolo fornita dal processore grafico e verranno presentati di seguito.

## 4.2 NVIDIA CUDA

CUDA (Compute Unified Device Architecture) è un modello di programmazione creato da NVIDIA come piattaforma per la computazione parallela. L'architettura CUDA include un linguaggio assembly (PTX) e un sistema di compilazione che rappresentano la struttura di base sulla quale è fondata l'intera computazione su GPU NVIDIA. La piattaforma CUDA consente diversi livelli di interazione ed è accessibile agli sviluppatori in diversi modi.

PTX è basato su librerie accelerate dalla GPU, come cuBLAS o cuFFT, permettendo un utilizzo semplice e il vantaggio di poter essere impiegato senza la necessità di specifiche conoscenze sul funzionamento della GPU. Questo tipo di librerie, infatti, realizzano versioni graficamente accelerate, quindi più performanti, di alcune utili funzioni. Lo svantaggio evidente di questa soluzione è la scarsa adattabilità alle esigenze del programmatore: le ottimizzazioni computazionali sono garantite solo su un ristretto numero di operazioni.

Un’altra soluzione consiste nell’impiego di direttive (come OpenACC). Questa tecnica prevede di inserire all’interno del codice particolari istruzioni che, interpretate dal compilatore, consentono di parallelizzarlo, utilizzando la potenza di calcolo dei processori grafici nel modo che il compilatore ritiene più efficace. Tale approccio è molto semplice da utilizzare ma, come il precedente, non consente un controllo completo sui meccanismi di elaborazione, delegando tutto il lavoro al compilatore.

L’ultima soluzione consiste nell’utilizzo di un linguaggio di programmazione general purpose esteso con istruzioni CUDA che permettono l’esecuzione su GPU. NVIDIA ha realizzato estensioni per i linguaggi C (CUDA-C/C++) e Fortran (CUDA-Fortran), ma altri partner hanno sviluppato wrapper che supportano, tra gli altri, i linguaggi Java, Python, Perl e Ruby. Gli sviluppatori che scelgono questa soluzione hanno quindi un accesso diretto alle caratteristiche hardware della GPU, come memoria condivisa, meccanismi di caricamento dati e core d’esecuzione. Questa tecnica è sicuramente la più complessa, ma è l’unica in grado di garantire allo sviluppatore il pieno controllo del processore grafico. La piattaforma CUDA è progettata per essere utilizzata esclusivamente con processori grafici prodotti da NVIDIA, caratteristica che ne limita l’impiego in misura considerevole. D’altra parte, questo vincolo è probabilmente il motivo per cui, sui dispositivi NVIDIA, CUDA è la soluzione che offre migliori performance: il limitato numero di dispositivi da supportare ha consentito la realizzazione di una piattaforma specializzata, mirata all’ottimizzazione di un numero ristretto di operazioni.


Figura 7 Scheda NVIDIA A100

## 4.3 OpenCL

OpenCL (Open Computing Language) è un framework per lo sviluppo di programmi in grado di lavorare su piattaforme eterogenee, che possono essere composte indifferentemente da CPU e da GPU realizzate da diversi produttori. Questa piattaforma è stata ideata da Apple, ma è stata sviluppata e mantenuta da un consorzio no-profit chiamato Khronos Group. OpenCL è la principale alternativa a CUDA per l'esecuzione di software su GPU, ma presenta un punto di vista diametralmente opposto: mentre CUDA fa della specializzazione il proprio punto di forza (prodotta, sviluppata e compatibile con NVIDIA), garantendo ottime prestazioni a scapito della portabilità, OpenCL propone una soluzione compatibile con la quasi totalità dei dispositivi presenti sul mercato. Il software scritto in OpenCL, infatti, può essere eseguito su processori (grafici e non) prodotti da tutte le maggiori industrie del settore, come Intel, NVIDIA, IBM, AMD.

OpenCL include un linguaggio per scrivere kernel basato su C99 (con alcune limitazioni), che consente di utilizzare in maniera diretta le potenzialità dell'hardware a disposizione, in maniera analoga a come avviene con CUDA-C o CUDA-Fortran. OpenCL mette a disposizione funzioni per l'esecuzione in ambiente altamente parallelo, primitive di sincronizzazione, qualificatori per le regioni di memoria e meccanismi di controllo per le diverse piattaforme d'esecuzione. La portabilità dei programmi OpenCL è però limitata alla possibilità di eseguire lo stesso codice su dispositivi diversi e non garantisce che le prestazioni siano ugualmente affidabili. Per ottenere le migliori prestazioni possibili, infatti, è fondamentale fare riferimento alla piattaforma d'esecuzione, ottimizzando il codice in base alle caratteristiche del dispositivo.

# Capitolo 5 La legge di Moore e il futuro

"La complessità di un microcircuito, misurata ad esempio tramite il numero di transistor per chip, raddoppia ogni 18 mesi (e quadruplica quindi ogni 3 anni)."

## 5.1 Prima legge di Moore

Questa legge è diventata il metro e l'obiettivo di tutte le aziende che operano nel settore come Intel e AMD e deriva da un'ipotesi di Gordon Moore, cofondatore di Intel. Egli, osservando la crescita delle prestazioni degli anni precedenti, ipotizzò che il numero di transistori nei microprocessori sarebbe raddoppiato ogni 12 mesi. Nel 1975 questa ipotesi trovò conferma, per poi essere modificata nella forma attuale nei primi anni Ottanta fino ai nostri giorni, viene riformulata alla fine degli anni Ottanta ed elaborata nella sua forma definitiva, ovvero che il numero di transistori nei processori raddoppia ogni 18 mesi.

Ecco alcuni esempi sulla prima legge di Moore:

| | | |
|---|---|---|
| Nel maggio del 1997 Intel lancia il processore Pentium II | Frequenza: 300 MHz | Numero di transistori: 7,5 milioni |
| Nel novembre del 2000, ovvero 42 mesi dopo il lancio del Pentium II, Intel mette in vendita il Pentium 4 | Frequenza: 1,5 GHz | Numero di transistori: 42 milioni |

Secondo la legge di Moore, dopo 18 mesi dal lancio del Pentium II, sarebbe stato possibile realizzare un processore contenente 15 milioni di transistor. Trascorsi altri 18 mesi (36 mesi dal lancio del Pentium II), il numero di transistor sarebbe raddoppiato rispetto ai 18 mesi precedenti passando da 15 a 30 milioni di transistor. Se fossero passati altri 18 mesi (54 mesi dal lancio del Pentium II) il numero di transistor sarebbe aumentato di 30 milioni, arrivando a 60 milioni. Dato che il Pentium 4 è stato lanciato tra 36 e 54 mesi dopo il lancio del Pentium II, il numero di transistori del Pentium 4 stimato dalla legge di Moore sarà compreso tra 30 e 60 milioni. Per calcolare una stima più precisa definiamo la funzione: $f(x) = 2^{((x/18))} \times 7.5$ che ci consente di stimare il numero di transistori dopo x mesi dal lancio del Pentium II.

Lancio sul mercato del Pentium II avente 7,5 milioni di transistor:

- 18 mesi dal lancio: f(18) = 15 milioni di transistori stimati dalla legge di Moore
- 36 mesi dal lancio: f(36) = 30 milioni di transistori stimati dalla legge di Moore
- 42 mesi dal lancio: f(42) = 37,8 milioni di transistor stimati dalla legge di Moore e lancio sul mercato del Pentium 4 con 42 milioni di transistori.

Come si può notare, in questo caso, la legge di Moore ha stimato una crescita addirittura inferiore rispetto a ciò che è effettivamente avvenuto.

Lo stesso ragionamento è valido per la frequenza del processore definendo $f(x) = 2^{((x/18))} \times 300$ che ci restituisce per f(42) proprio 1,5 GHz, la frequenza del Pentium 4.

## 5.2 Limiti della prima legge di Moore:

I limiti della prima legge di Moore riguardano solo il raggiungimento dei limiti fisici imposti per la riduzione delle dimensioni dei transistor, e quindi della scala di integrazione, al di sotto dei quali si genererebbero effetti 'parassiti' indesiderati di natura quantistica nei circuiti elettronici. Tali limiti sarebbero pressapoco già stati raggiunti con la generazione dei processori Pentium al di sopra del quale l'unico modo possibile e praticabile per aumentare le prestazioni di calcolo è rappresentato dalla tecnologia multicore, ovvero dall'accoppiamento in parallelo di più processori come avviene peraltro nei supercalcolatori dei centri di calcolo.

Tali capacità di integrazione e quindi di calcolo rendono possibile l'utilizzo di applicazioni informatiche sempre più complesse quali ad esempio quelle legate all'industria dell'intrattenimento quali i videogiochi oppure della computer grafica, mentre semplici applicazioni di calcolo matematico, se si esclude il calcolo scientifico ad alte prestazioni, sono soddisfacibili anche con meno risorse di calcolo.

## 5.3 Seconda legge di Moore:

Gordon Moore affermava che:

“Sarebbe molto più economico costruire sistemi su larga scala a partire da funzioni minori, interconnesse separatamente. La disponibilità di varie applicazioni, unita al design e alle modalità di realizzazione, consentirebbe alle società di gestire la produzione più rapidamente e a costi minori.”

Non si tratta certamente di considerare il logaritmo sulla memoria dei chip, già noto al fondatore di Intel, ma dell'implicita enunciazione di un'altra importante legge, ovvero quella che riguarda l'efficienza dei dispositivi elettronici ed il loro costo effettivo. Come nel caso della prima, si tratta di un accordo tra fattori diversi. Ma non fu Moore a trattare questi argomenti.

Arthur Rock, uno dei primi investitori nella Intel, osservò che il costo dei macchinari per la fabbrica di cui era azionista raddoppiava ogni quattro anni circa. Da qui un primo principio su cui fondare una nuova legge:

"Il costo delle apparecchiature per fabbricare semiconduttori raddoppia ogni quattro anni".

In seguito, Moore integrò definitivamente la sua legge originaria formulandone una seconda:

"il costo di una fabbrica di chip raddoppia da una generazione all'altra"

Egli fece quest'affermazione in base all'osservazione della dinamica dei costi legati alla costruzione delle nuove fabbriche di chip: i costi erano passati da 14 milioni di dollari nel 1966 a un miliardo di dollari nel 1996. Questi erano quindi cresciuti a un ritmo superiore rispetto all'incremento di potenza dei chip previsto dalla prima legge. La proiezione di questi costi indicava che nel 2005 una fabbrica di chip sarebbe costata 10 miliardi di dollari. L'implicazione di questo primo andamento è che il costo per transistor, oltre a smettere di diminuire, sarebbe destinato ad aumentare. Moore aveva avvertito che la validità della sua prima legge stava per giungere a termine.

Da notare però che nello stesso periodo Moore in qualche modo smentiva sé stesso dato che gli sviluppi realizzati nella litografia (anche presso l'Università del Texas) consentirono dei risparmi di costo e dei miglioramenti di qualità dell'output che confermavano la validità della prima legge per almeno un altro decennio. Da considerare è anche un altro aspetto che emerge quando si cita la legge di Moore: l'accentuazione che generalmente viene data al vantaggio competitivo offerto dalla tecnologia dei processori e dalle implicazioni fornite anche dalle "code" di questa tecnologia.

Poiché i processori d'avanguardia aumentano di potenza a parità di prezzo, i processori della generazione precedente, la cui potenza rimane fissa, calano di prezzo. Come osservò Karlgaard (1998):

"il corollario è che nel 2008 i chip Pentium II e PowerPC costeranno circa 75 cent".

Significa che sarà conveniente utilizzare questi chip negli elettrodomestici, negli autoveicoli e in tutte le applicazioni ad ampia diffusione, insieme a questo è giusto considerare l'uso non destinato al personal computing di processori a 32 bit e si nota come delle unità ad Architettura MIPS dei modem ADSL con molta elettronica periferica a bordo hanno un costo (in quantità) ben inferiore al mezzo dollaro. Di conseguenza, occorre effettivamente analizzare anche il ciclo del valore e dei volumi del prodotto finale.

Tutte le innovazioni tecnologiche e il miglioramento della qualità dei materiali che hanno reso possibile il processo di scala dei dispositivi hanno comportato, però, investimenti sempre crescenti in apparecchiature: da queste osservazioni si può comprendere il perché di un'ulteriore interpretazione della seconda legge di Moore:

"L'investimento per realizzare una nuova tecnologia di microprocessori cresce in maniera esponenziale con il tempo".

Ovviamente, per incrementare le prestazioni, occorrono sempre maggiori studi, ricerche e test. Per aumentare il numero di transistor all'interno del processore, senza aumentare la dimensione del processore stesso, occorrono dei componenti sempre più piccoli, quindi nuovi materiali che permettano questo risultato. L'aumento delle prestazioni comporta dei test, sia per provare la resistenza dei materiali, sia per l'affidabilità stessa del processore. Tutto questo ovviamente comporta delle spese che la casa produttrice deve affrontare se vuol avere un prodotto funzionante e funzionale.

Ogni nuova linea pilota richiede, quindi, investimenti (e coinvolge ricercatori) paragonabili con quelli degli acceleratori di particelle o dell'esplorazione spaziale. Anche se l'industria microelettronica spende circa il 20% del proprio fatturato in nuove fabbriche e il 12-15% in ricerca e sviluppo, la crescita degli investimenti richiesti per una nuova linea pilota tende a rappresentare una porzione, sempre più alta, del fatturato, con alcune implicazioni economiche rilevanti:

- riduzione nel numero di società che si possono permettere linee pilota avanzate (Intel, AMD, NVIDIA);
- fenomeni di associazione di società diverse per condurre la ricerca in comune;
- crescita dei rischi connessi ad un investimento sbagliato, che colpisce, soprattutto, le società che sviluppano le attrezzature di produzione nel settore della microelettronica.

## 5.4 Le nuove leggi:

Si riteneva che le leggi di Moore non sarebbero più potute essere sostenuta dopo il 2020, in quanto già dopo il 2000 risultava palese che le prestazioni dei microprocessori non fossero migliorate in modo significativo, se non grazie a nuove strutture architetturali delle stesse.

“Possiamo aumentare il numero dei transistor e dei core di quattro volte ogni tre anni. Facendo lavorare ogni core leggermente più lentamente e perciò in maniera più efficiente, possiamo più che triplicare le prestazioni mantenendo lo stesso consumo totale”. Questa è la legge enunciata da Bill Dally, Capo scienziato NVIDIA, portabandiera di una nuova era nell'evoluzione dei microprocessori, con l’obiettivo di passare dal calcolo seriale al calcolo parallelo.

Come abbiamo visto nei capitoli precedenti questa era che è già iniziata con l’utilizzo in svariati ambiti delle GPU e del loro modo di processare dati.

# Bibliografia


- GPU: processori manycore di Annalisa Massini


# Sitografia


- www.intelligenzaartificiale.it
- www.benchmarkmagazine.com
- www.lescienze.it
- www.spremutedigitali.it
- www.ibm.it
- www.gazzetta.it