ARCHITETTURE DEI CALCOLATORI
l'approccio GPU

Mariagiovanna Sami

Una prima definizione di GPU

- L'elaboratore grafico (Graphic Processing Unit – GPU): dispositivo programmabile progettato per eseguire con altissime prestazioni programmi di sintesi grafica… ma non solo!
- Applicazioni considerate: caratterizzate da numeri molto alti di thread (anche più di 10.000…) *tutti essenzialmente copie (istanze) di uno stesso thread* che possono essere eseguiti in parallelo, con una predominante reciproca indipendenza per quanto riguarda i dati.

Una prima definizione di GPU

- Con un carico di lavoro di questo tipo, si cambia l'approccio al progetto:
  - *Non occorre ricorrere a esecuzione speculativa:* c'è comunque un gran numero di thread che aspettano di essere eseguiti!
  - Non si ricorre quindi né a esecuzione speculativa dei salti, né a predizione dei salti
  - Le cache servono per fornire banda; si ricorre a multi-threading per mascherare la latenza di memoria!
- Come è possibile lanciare e gestire un così gran numero di thread?
- E se al momento di una branch le diverse istanze del thread divergono? Se accedono in modo casuale a blocchi/banchi di memoria?

Una prima definizione di GPU

- L'architettura privilegia la presenza di un numero molto elevato di processori relativamente semplici – le alte prestazioni si ottengono dall'esecuzione parallela di moltissimi thread piuttosto che dall'esecuzione ottimizzata di un unico thread!
- Ci si trova in presenza di architetture *many-core* (multi-core: da due a sedici CPU su un chip; many-core: centinaia di CPU su un chip);
- Architettura di riferimento: Nvidia Tesla e Fermi (architettura sottostante al paradigma di programmazione CUDA).

Il modello architetturale

- La struttura multiprocessore è definita “SIMT” cioè “*Single-Instruction, Multiple-Thread”*
- Che cosa vuol dire? In sostanza, un’estensione del concetto SIMD.
- Il parallelismo a livello di thread rispecchia una *gerarchia di thread* (i thread vengono raggruppati in *warp* a loro volta strutturati in *blocchi* che costituiscono una *griglia).* In corrispondenza, si vede una gerarchia di strutture hardware.

Il modello architetturale

I blocchi di thread vengono eseguiti da *Streaming Multiprocessors* (*SM*):

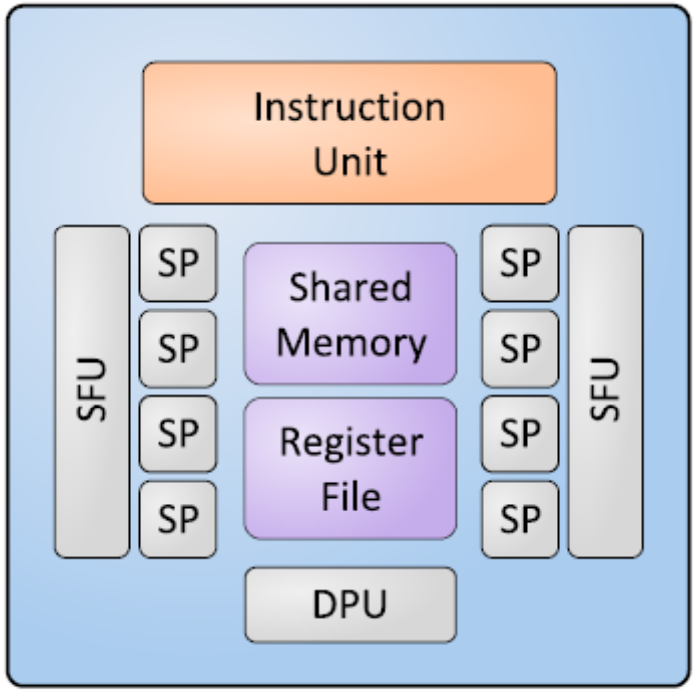


Fig. 1: Streaming Multiprocessor with 8 Scalar Processors Each

Uno Streaming Multiprocessor assomiglia a un processore vettoriale di larghezza 8 (contiene 8 *unità scalari*) che opera su vettori di larghezza 32.

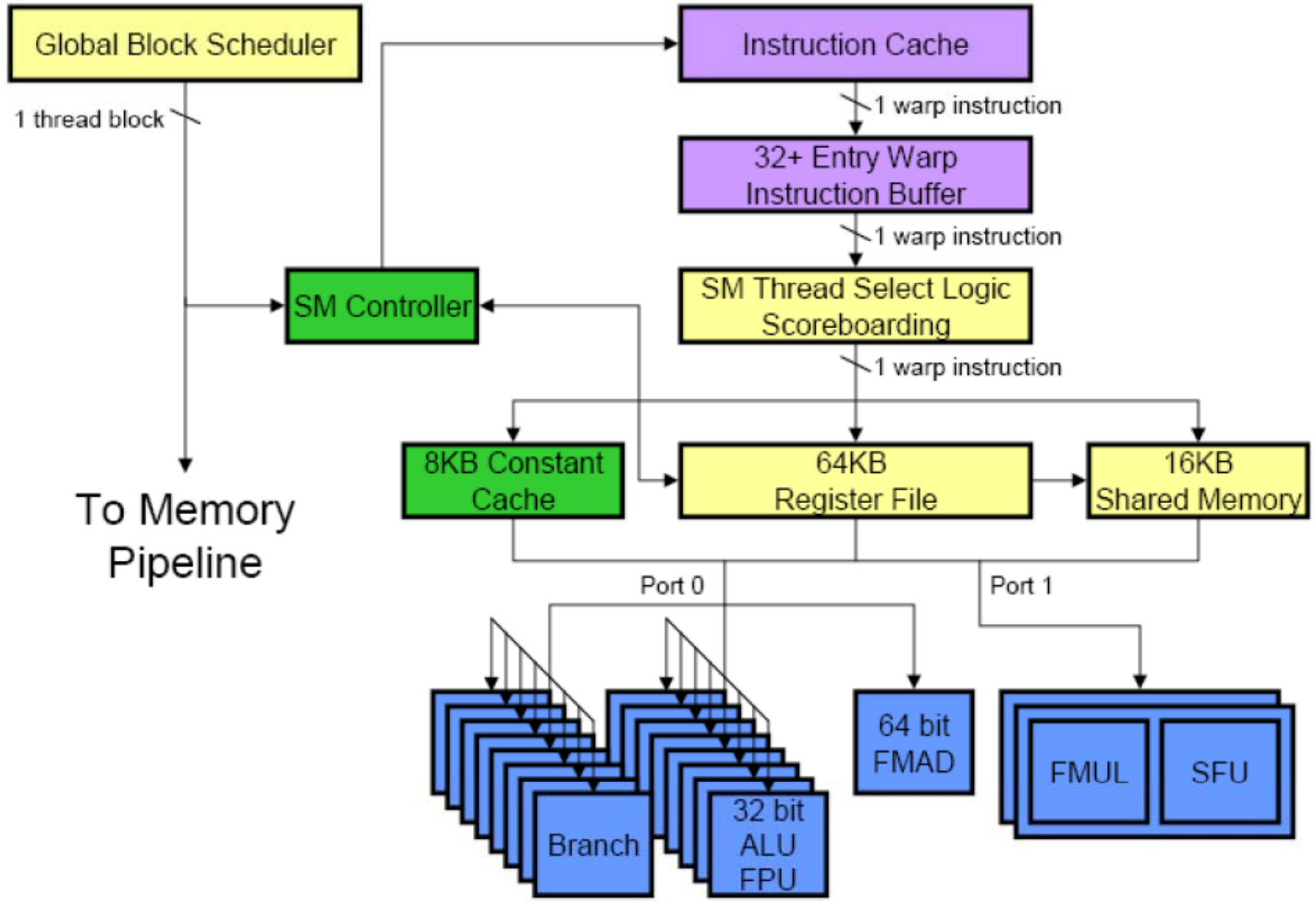
Il modello architetturale

- Ogni SM contiene tre diversi tipi di unità di esecuzione:

1. Otto processori scalari (SP) che eseguono istruzioni aritmetico/logiche in virgola mobile e in virgola fissa
2. Due Unità Funzionali Speciali (SFU) che eseguono varie funzioni trascendenti e matematiche (seno, coseno, etc.) oltre alla moltiplicazione in virgola mobile (in singola precisione)
3. Un'unità di doppia precisione (DPU) che esegue operazioni aritmetiche su operandi in virgola mobile di 64 bit.

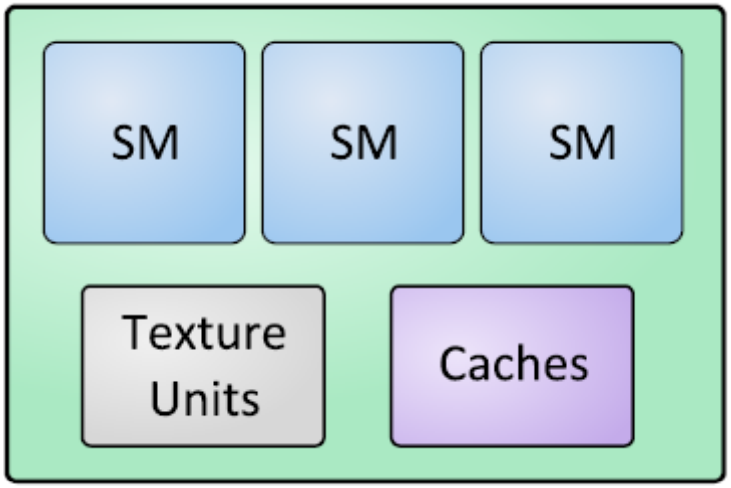
Il modello architetturale

- Lo Streaming Multiprocessor in dettaglio:



Il modello architetturale

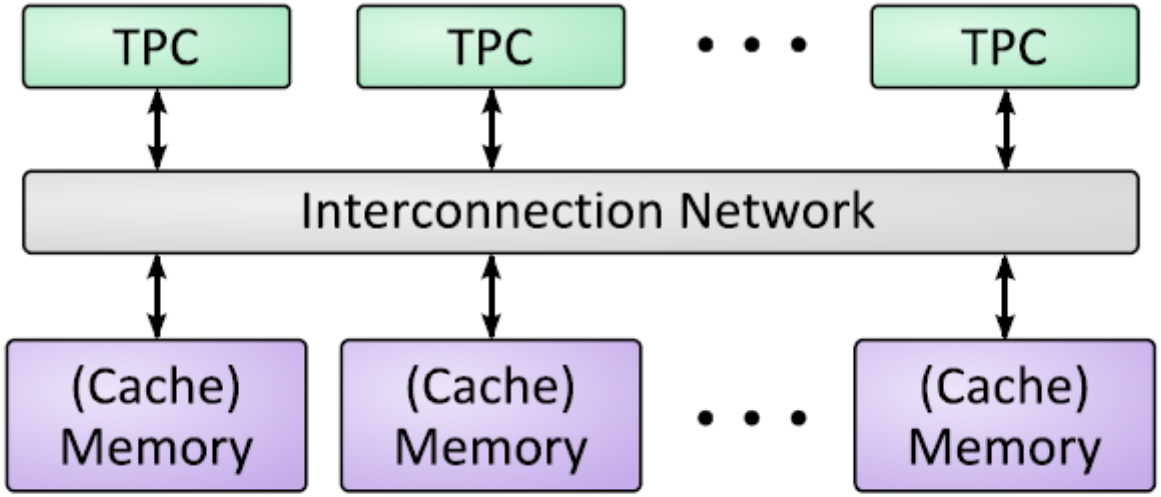
- Gruppi di SM appartengono a *Thread Processing Clusters* (TPC) che contengono anche altre risorse hardware tipiche delle applicazioni grafiche e in genere non visibili al programmatore.



UN TPC contenente tre SM

Il modello architetturale

La GPU è costituita dall'insieme dei TPC, dalla rete di interconnessione, dai controllori della memoria esterna



Il flusso di esecuzione

- Unità base del flusso di esecuzione: *warp* – nel GT 200, un insieme di 32 thread eseguito a gruppi di otto su otto processori scalari (cioè su un SM);
- Tutti i thread di un warp eseguono la stessa istruzione in maniera “lockstepped” (sincronizzazione forzata) ma ogni thread può seguire diramazioni diverse in corrispondenza di salti condizionati; più precisamente:
- I thread in un warp partono tutti dallo stesso indirizzo (“lo stesso PC”!);
- Un warp che esegue un salto condizionato aspetta che l’indirizzo obiettivo venga calcolato per ogni thread nel warp;

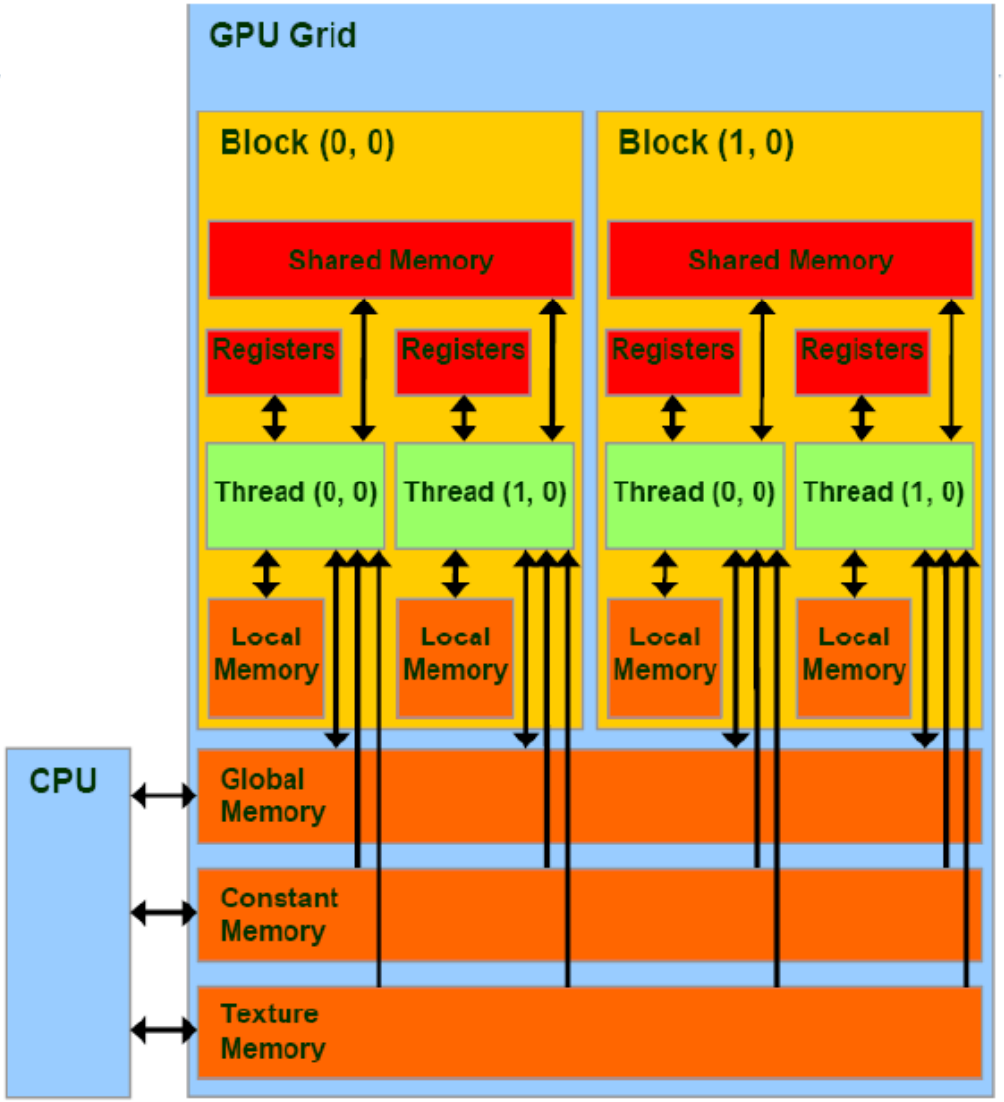
Il flusso di esecuzione

- Se tutti i thread nel warp seguono lo stesso percorso non c'è problema…
- quando i thread divergono in conseguenza di un salto condizionato dipendente dai dati:
  - il warp esegue *in maniera seriale* ogni percorso che è stato preso partendo dal punto di diramazione, disabilitando di volta in volta i thread che non sono su quel percorso. I thread entro il warp che seguono lo stesso percorso vengono eseguiti in parallelo.
  - Quando l'esecuzione dei percorsi conseguenti a una divergenza termina, i thread *riconvergono* in uno stesso percorso di esecuzione. Essenzialmente questo aspetto caratterizza la modalità SIMT rispetto alla "classica" modalità vettoriale.
- Una divergenza si presenta solo all'interno di un warp – warp diversi vengono eseguiti indipendentemente l'uno dall'altro.

Il flusso di esecuzione

- Load e Store vengono generate internamente agli SM (calcolo dell'indirizzo come somma registro base + spiazzamento e traduzione da indirizzo virtuale a indirizzo fisico);
- Load e Store vengono lanciate un warp per volta ed eseguite in gruppi di mezzo warp (16 accessi a memoria per volta);
- La sincronizzazione fra i diversi warp all'interno di un blocco viene eseguita mediante un meccanismo di barriera;

Il modello di memoria



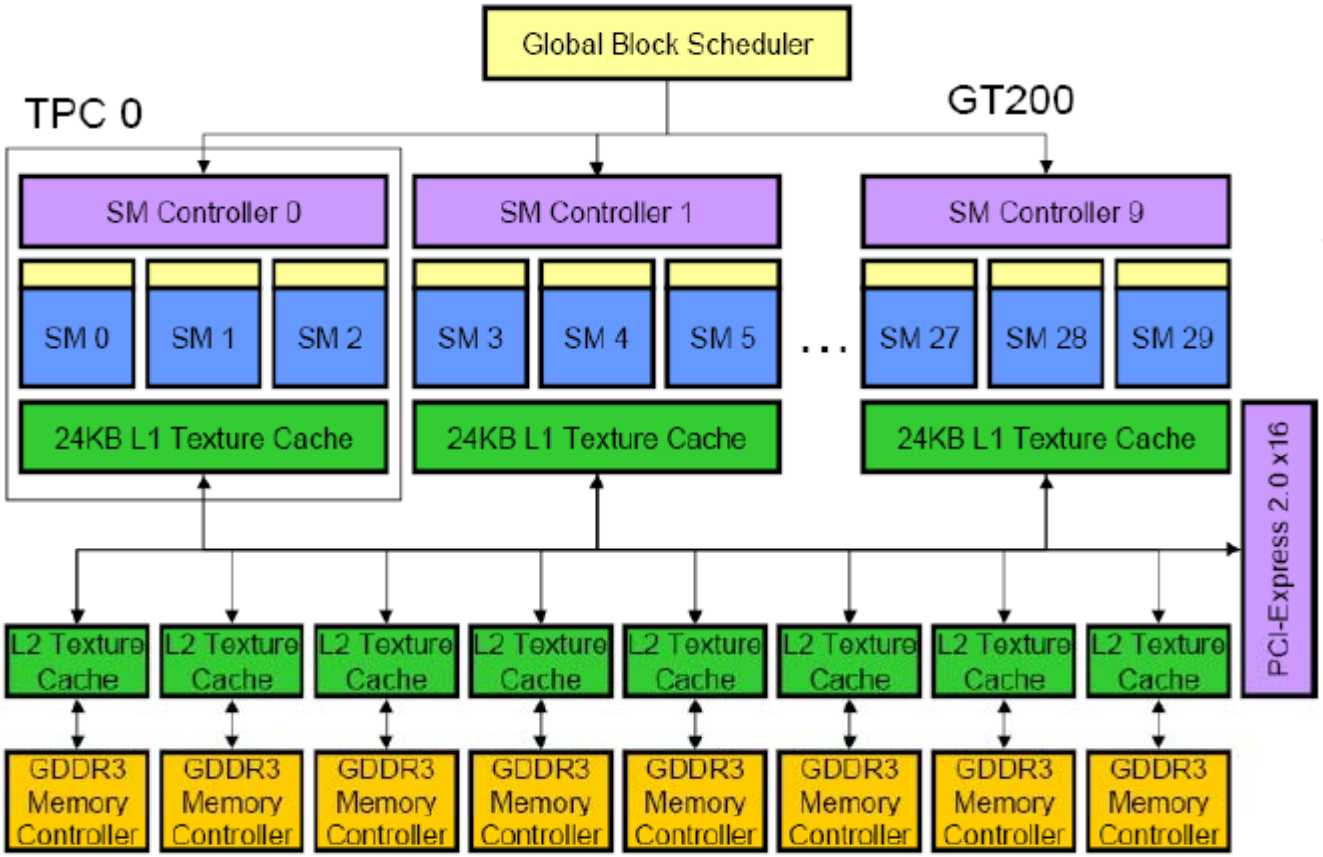
- Un modello *logico*, non fisico!
- Blocco: = 64 thread
- 16.384 registri di 32 bit;
- Memoria condivisa che non può essere trasferita in cache: relativa a ogni SM, usata dai thread di un blocco per condividere dati con altri thread dello stesso blocco. (16 KB per blocco).

Il modello di memoria

- Memoria globale: accessibile da tutti i thread in esecuzione, anche se appartenenti a blocchi diversi;
- Texture memory: cached, a sola lettura, tipica dell'elaborazione grafica
- Constant memory: costituita da due segmenti, uno accessibile all'utente, l'altro usato per le costanti generate dal compilatore.
- Tre livelli di cache istruzioni (L1 di 4 KB in ogni SM, L2 di 8 KB allocata al TPC, L3 globale)

L'evoluzione

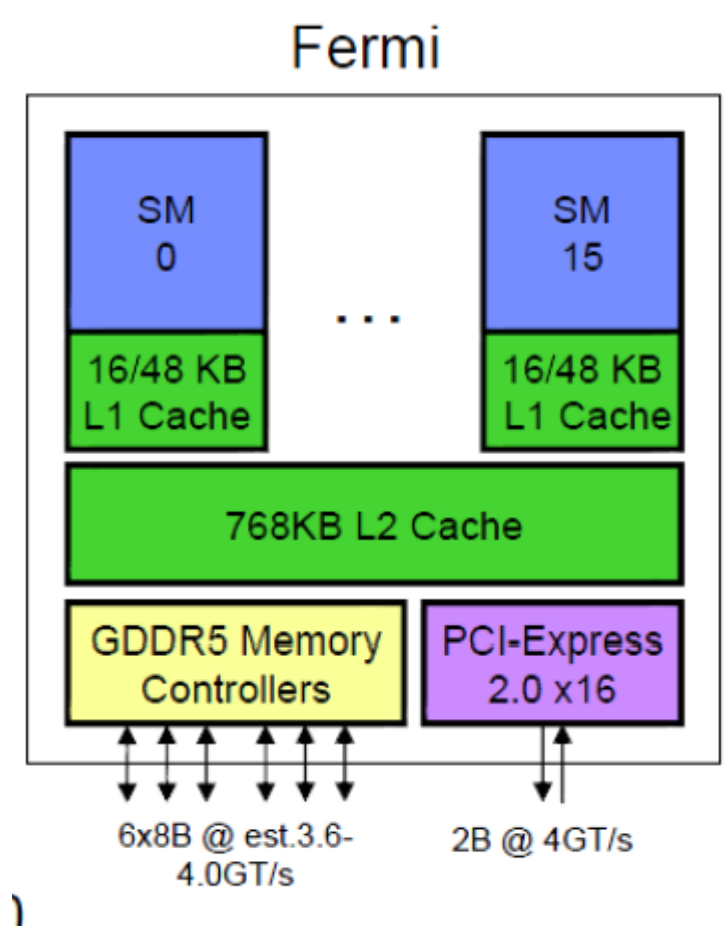
- Architettura GT 200 ("TESLA")



L'evoluzione

- Il successore ("FERMI"):
- Più fortemente orientato

al calcolo generale.



CPU + GPU

- Una GPU *non* è un elaboratore di uso generale: occorre prendere in considerazione un sistema “CPU + GPU” in cui un calcolatore general purpose (es., un PC) “scarica” attività sulla GPU;
- Un problema fondamentale nel progetto di un’applicazione: stabilire quali (e quante) parti dell’applicazione si prestano meglio ad essere eseguite dalla GPU;

CPU + GPU

- Modello di programmazione CUDA (Nvidia): un'estensione del C che include:
  - Codice "seriale" che verrà eseguito dalla CPU
  - Codice "parallelo" che verrà eseguito dalla GPU.
- CUDA: soluzione proprietaria Nvidia.
- Recente standard industriale "aperto": OpenCL orientato a calcolo task-parallel e data-parallel su sistemi eterogenei per uno spettro di CPU, GPU, DSP

Fonti.

(Oltre al materiale Nvidia):

- H. Wong, M.M. Papadopoulou, M. Sadooghi-Alvandi, and Andreas Moshovos: “Demystifying GPU Microarchitecture through Microbenchmarking”, *Proc. 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*
- Anton Lokhmotov, Paul H J Kelly: “Graphics and manycore”, *Lecture notes (Univ. of Cambridge), 2010*
- J. E. Stone, D. Gohara, G. Shi: “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems”, *Computing in Science and Engineering, 2010*