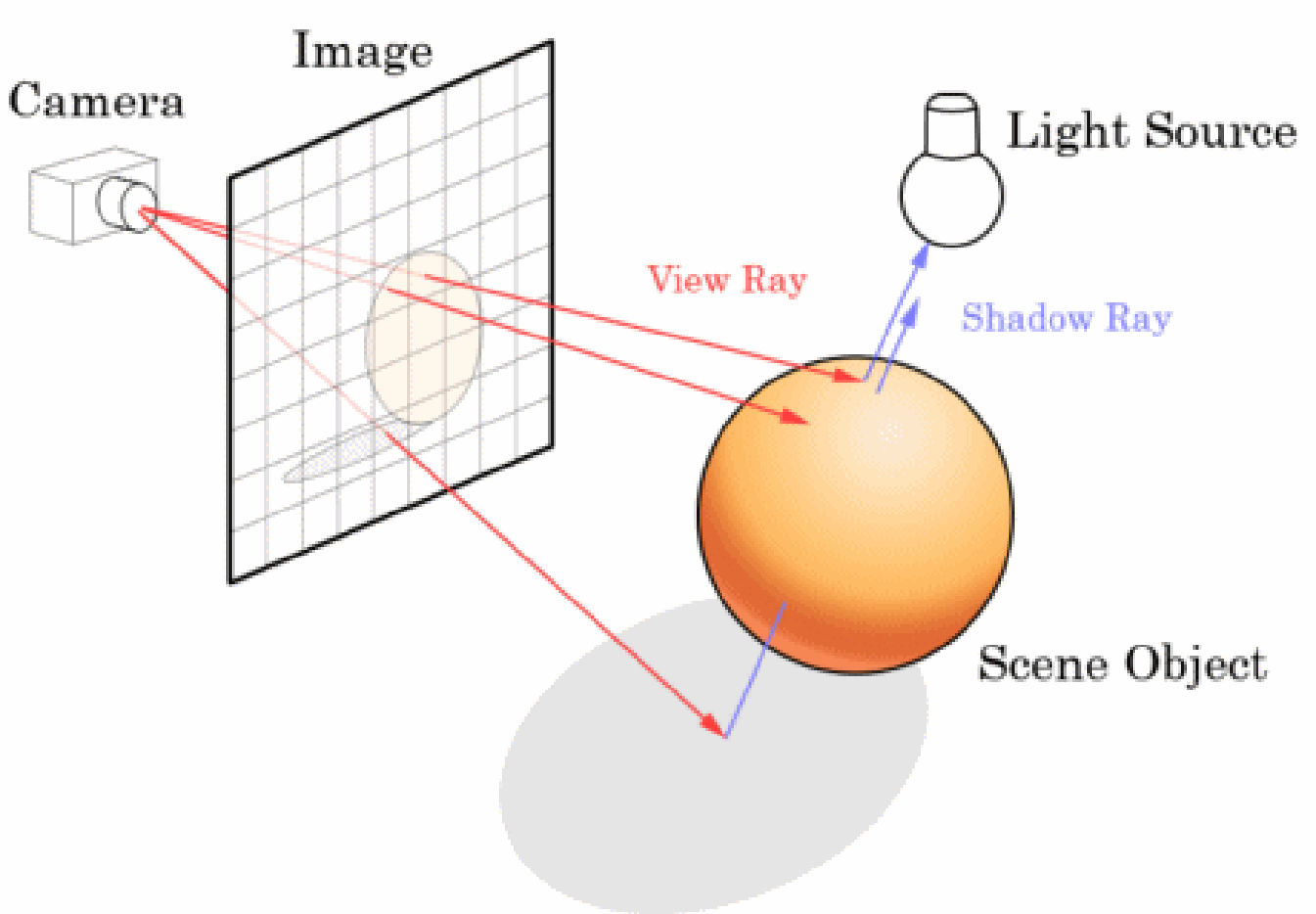
GPGPU Computing e CUDA

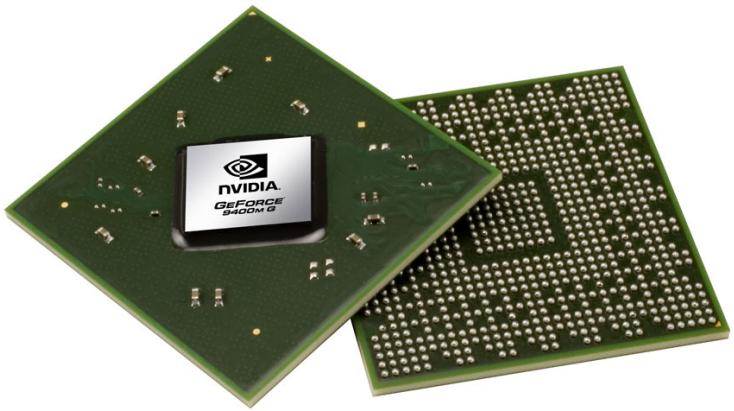
Argomenti

Argomenti

General-Purpose computation on GPU



General-Purpose computation on GPU



General-Purpose computation on GPU

- A partire dal 2003, cioè dalla introduzione delle DIRECTX 9, le GPU potevano essere utilizzate per calcoli matematici, a patto che i dati del problema venissero resi sotto forma di "immagini": si parlava di texture, shaders, memoria del framebuffer.
- NVIDIA, con l'introduzione della scheda GeForce 8800 (chip G80) nel novembre 2006 fornisce agli sviluppatori il framework CUDA grazie al quale i dati dei problemi matematici possono essere gestiti con i tipi primitivi del C: `double`, `float`, `int`, ...
- Con la seconda generazione del giugno 2008 (chip GT200), es. GTX280, NVIDIA supporta a pieno il calcolo in virgola mobile in doppia precisione.
- La terza generazione (chip Fermi) del 2010 permette a NVIDIA di lanciare una GPU il cui obbiettivo principale è il calcolo parallelo.

Risorse:

http://gpgpu.org Fermi_Architecture_Whitepaper.pdf

General-Purpose computation on GPU

Argomenti

Framework CUDA

CUDA: Compute Unified Device Architecture

- architettura di calcolo "massivamente parallela"
  - grande quantità di cores (centinaia su un singolo chip)
  - memoria condivisa (on-chip) e accesso ad una memoria globale dedicata (decine di GB di DDR5)
  - hardware di calcolo dedicato: MAD (Multiply And Add), DP Double Precision floating-point, SFU Spcual Function Units, ecc...
- modello di programmazione parallela scalabile, toolkit di programmazione completo
  - CUDA come estensione di C/C++
  - compilatore, debugger e profiler (per Win, Linux, Mac), distribuito gratuitamente da NVIDIA

Estensioni per altri linguaggi: Python, Matlab, FORTRAN.

CUDA e computing

Alcuni esempi di risorse disponibili per CUDA, distribuite da NVIDIA:

- cuBLAS
  Basic Linear Algebra Subprograms, quasi tutte le funzionalità delle BLAS sono state reimplementate sfruttando CUDA.
- cuFFT
  libreria simile a FFTW, offre algoritmi per il calcolo della FFT (Fast Fourer Transform).
- THRUST
  libreria ispirata alla STL (Standard Template Library). Libreria open-source di template C++: gestione di reduction, ordinamento, liste, per le architetture parallele a memooria condivisa (CUDA e OpenMP). Inclusa nel pacchetto CUDA dalla versione 4.0.

Argomenti

Pro

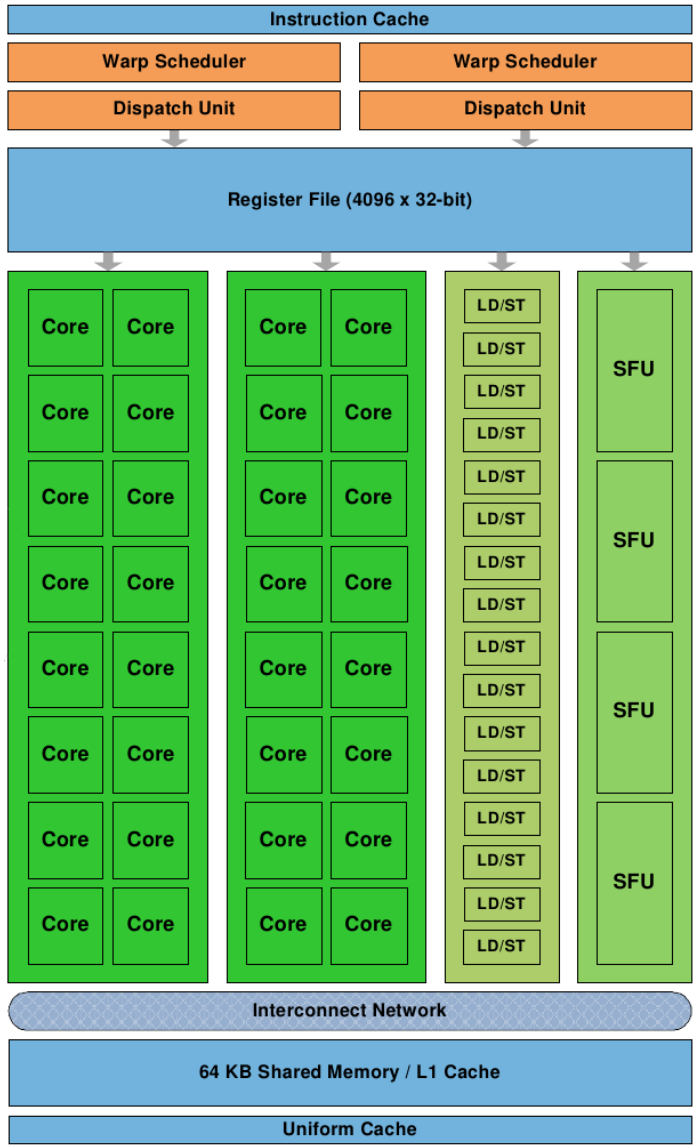
- Linguaggio di programmazione relativamente semplice
  - alcune estensioni al linguaggio C/C++
- Esempi di codice sorgente ampiamente disponibili
  - distribuite da NVIDIA (SDK) e altri
- Particolarmente efficace per semplici operazioni
  - es: calcoli semplici su una discreta mole di dati (approccio SIMD-like, chiamato SIMT da NVIDIA)
- Costo limitato per ogni lancio di thread
  - l'architettura è capace di sopportare centinaia o migliaia di threads che girano in parallelo.

Contro

- Per ottenere codice ottimizzato, si ha bisogno di studiare l'architettura fisica
  - gestione della memoria, gestione dei thread...
- Scarse prestazioni in caso di threads "divergenti"
  - i threads sono gestiti a gruppi: threads che eseguono istruzioni non appartenenti alla stessa riga di codice sono eseguiti serialmente.
- Non è indicata per problemi di alte dimensioni:
  - se i dati non possono essere contenuti interamente nella memoria del dispositivo, gli speedup ottenuti dall'approccio CUDA possono essere limitati dalle copie: host-to-device e device-to-host.

Argomenti

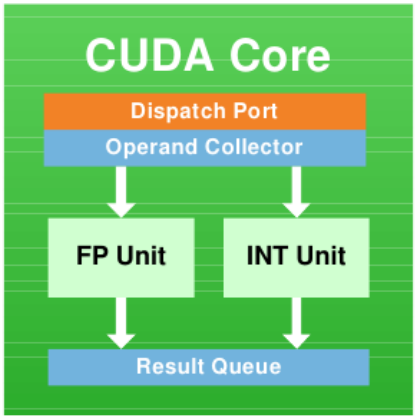
Streaming Multiprocessor



Streaming Multiprocessor (SM):

- 32 processori (Cores)
- 16 unità load/store per il calcolo di indirizzi di memoria (LD/ST)
- 4 unità per il calolo di funzioni trigonometriche, inverso, radice quadrata (SFU).
- 2 unità di controllo (Warp Scheduler)

Core



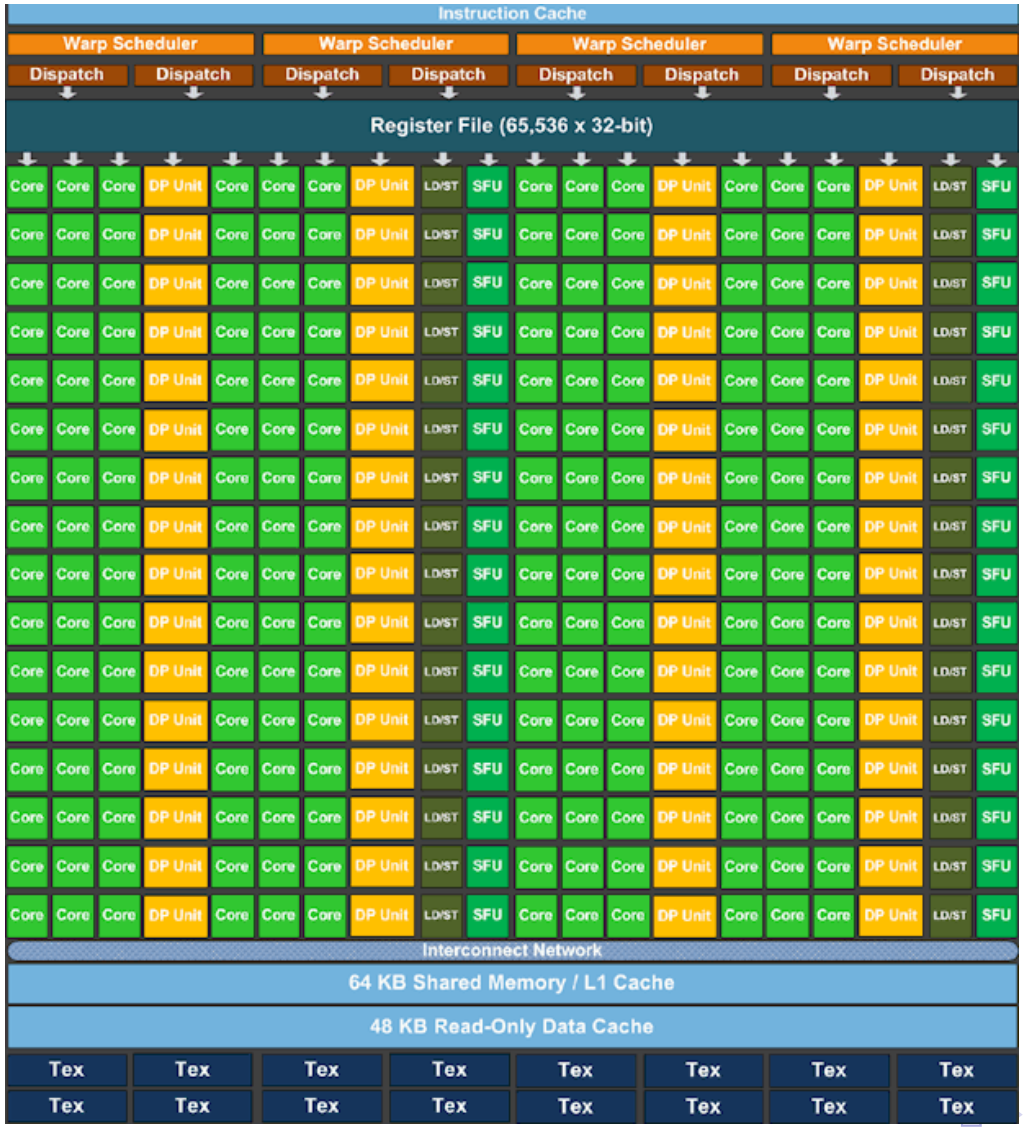
Core:

- 1 unità aritmetico-logica (ALU)
- 1 unità fp a singola precisione (FPU)

- I thread sono raggruppati in warp (gruppi di 32 threads), e controllati simultaneamente da un unico warp scheduler.
- Per il calcolo in doppia precisione, ogni operazione coinvolge una coppia di Core.

Argomenti

Kepler

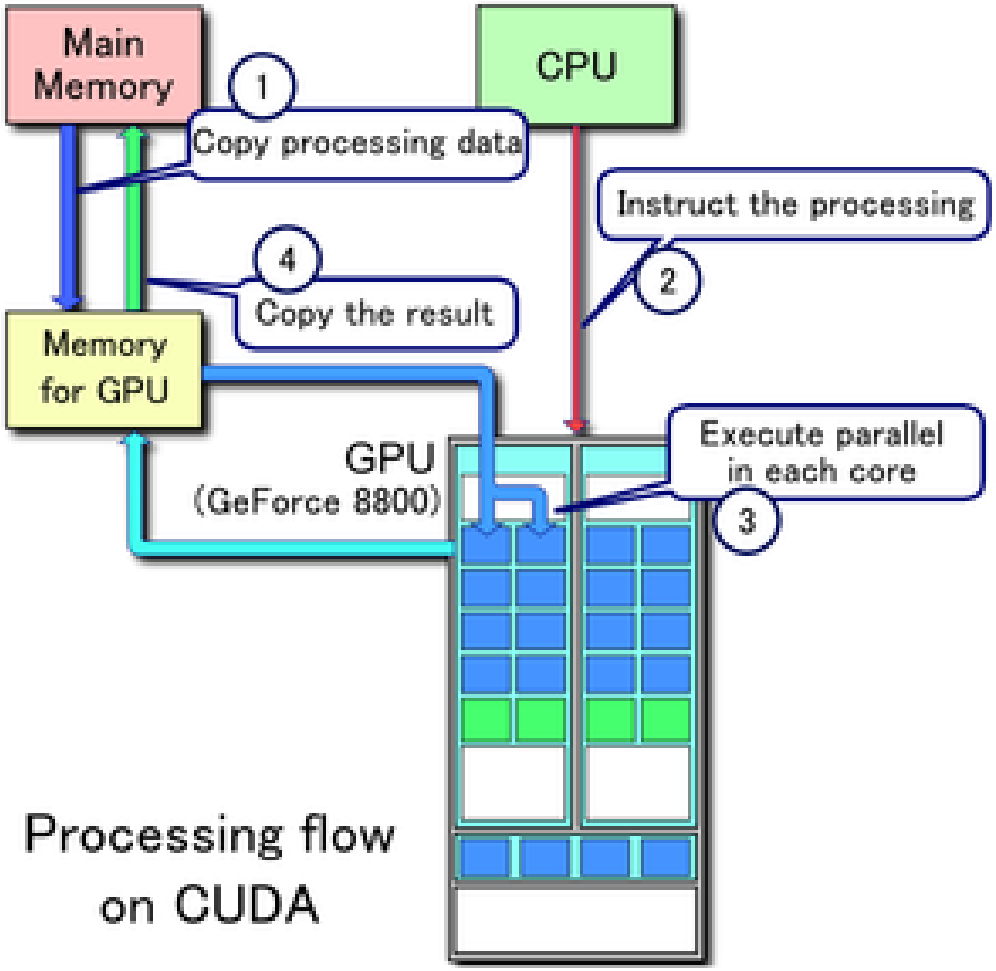

Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex

Workflow

Struttura di un semplice programma CUDA:

1. copia dei dati dalla memoria host (PC) alla memoria device (scheda GPU),
2. lancio di un Kernel CUDA,
3. copia dei risultati dal device all'host.

Workflow


Main Memory
CPU
1
Copy processing data
Instruct the processing
2
4
Copy the result
Memory for GPU
GPU
(GeForce 8800)
Execute parallel in each core
3
Processing flow on CUDA

Kernel CUDA

Un Kernel cuda è una speciale funzione C/C++ che viene invocata dall'host e viene eseguita dal device:

CUDA Kernel

```
__global__ void my_kernel (...){
            /* Calcoli svolti sul device */
}
```

Un Kernel cuda:

- ritorna `void`
- accetta parametri come le classiche funzioni C
- è dichiarato con l'estensione CUDA `__global__`

Estensioni CUDA per la dichiarazione di funzioni

Definizione			Chiamante	Esecutore
__device__	float	DeviceFunc()	device	device
__global__	void	KernelFunc()	host	device
__host__	float	HostFunc()	host	host

- Una funzione di tipo `__host__` è la tradizionale funzione C, questa è l'estensione di default.
- Una funzione può essere dichiarata sia `__host__` che `__device__`: il compilatore in questo caso genera due versioni della stessa funzione, una per l'host, l'altra per il device.
- Una funzione di tipo `__host__` o `__device__` può avere valore di ritorno.