Architetture GPGPU

- Architetture per il calcolo parallelo
- Modello CUDA
- Un esempio pratico

Architetture per il calcolo parallelo

- **SIMD** sta per «Single Instruction Multiple Data»: la stessa istruzione viene eseguita su un vettore di dati
- **SMT** sta per «Simultaneous MultiThreading»: molti thread differenti operano in parallelo
- **SIMT** «Single Instruction Multiple Threads» - introdotto da NVIDIA, si colloca nel mezzo

SIMD < SIMT < SMT

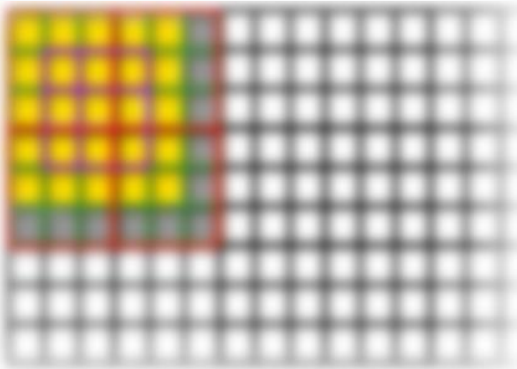
Un vantaggio del calcolo parallelo con SIMT

Principio base:

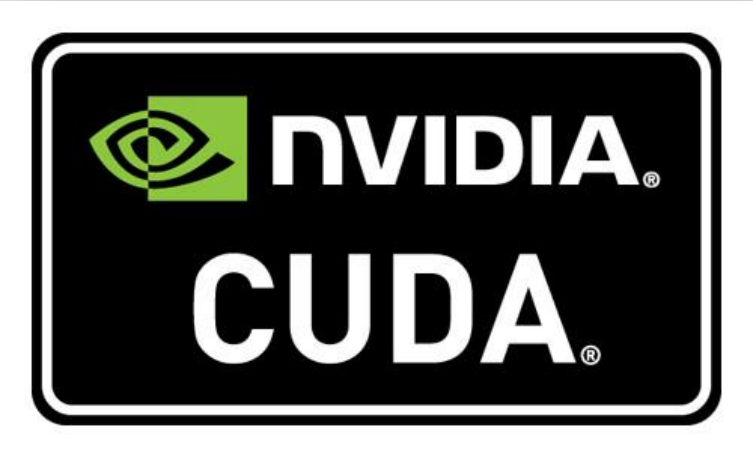
Se un thread richiede un trasferimento dati, la GPU seleziona un altro thread da eseguire

Se ci sono tantissimi thread da eseguire, la probabilità che TUTTI siano bloccati su un trasferimento dati è estremamente bassa.

Questo equilibrio tiene la GPU sempre occupata durante i trasferimenti di memoria (latency hiding)



CUDA – Compute Unified Device Architecture



CUDA (Compute Unified Device Architecture) è una piattaforma e modello di programmazione per il calcolo parallelo su GPU che sfrutta l'architettura SIMT

CUDA è stata sviluppata da NVIDIA e concepita per l'utilizzo su schede proprietarie ma esistono anche piattaforme alternative come openCL.

Getting CUDA

Per utilizzare CUDA sono necessari:

- Una GPU NVIDIA che supporti CUDA (https://developer.nvidia.com/cuda-gpus)

Una versione supportata di Linux, gcc e relativa toolchain

Una versione supportata di OS X, gcc/clang e relativa toolchain installati con Xcode

Una versione supportata di Windows, Visual Studio

- Il Toolkit CUDA (https://developer.nvidia.com/cuda-downloads)

Cos'è un Kernel CUDA

Un kernel CUDA, in termini estremamente semplici,

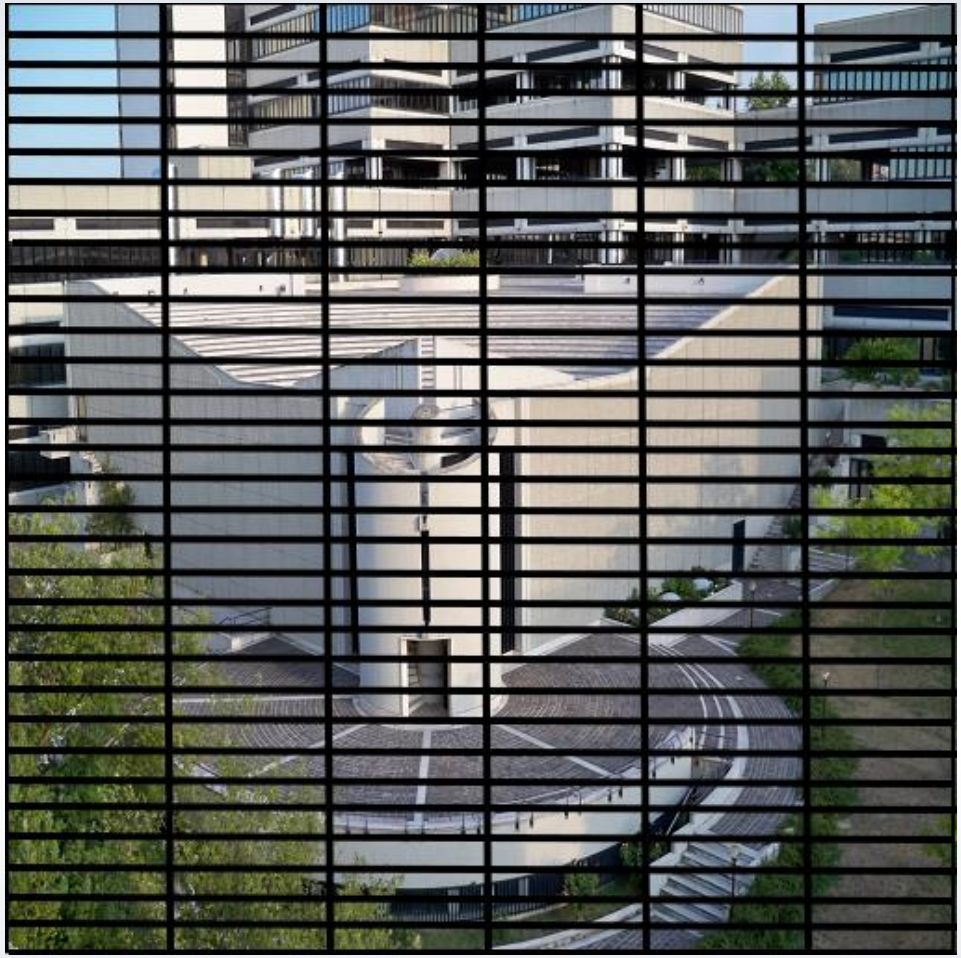
è una funzione eseguita sulla GPU in parallelo da moltissimi thread

```
// La parola chiave __global__ non è standard del C++ ma una estensione
// NVIDIA per indicare che questo kernel è globale e può essere chiamato
// da host (da codice che gira sulla CPU) ed eseguito sul device (GPU)

__global__ void kernelCUDA (char *a, int *b) {
    a[threadIdx.x] += b[threadIdx.x];
}
```

Blocchi di threads

CUDA fornisce un meccanismo di astrazione e raggruppamento dei threads che eseguono un kernel in **una griglia di blocch**i. Questo consente di suddividere un problema in sotto-problemi e di far collaborare i thread di un blocco alla sua risoluzione



Un esempio pratico

GitHub

http://marcodiiga.github.io/gp2univpm/

Gaussian Blur convolution (CPU / GPU)

Parallelizzabilità di un algoritmo

Non tutti gli algoritmi si prestano ad un guadagno prestazionale se eseguiti sulla GPU

Il semplice algoritmo di convoluzione sul repository, ad esempio, si presta molto bene alla parallelizzazione

Benchmark hardware

i7-4770 @ 3.40 GHz
8 GB RAM
GeForce GTX 650 Ti
CUDA 6.5

Errori comuni nella programmazione CUDA

- Ridurre al minimo i trasferimenti device <-> host (i trasferimenti sul bus PCI-e sono molto lenti)
- Ridurre la divergenza dei thread
- Usare tecniche di collaborazione fra i threads di un blocco per aumentare le prestazioni (i.e. reduction)
- Mantenere alta la coalescenza delle letture (se i threads non leggono singolarmente dalla memoria, le prestazioni saranno migliori)
- Altri consigli su http://bit.ly/WavePDEGPGPU

Buon lavoro!