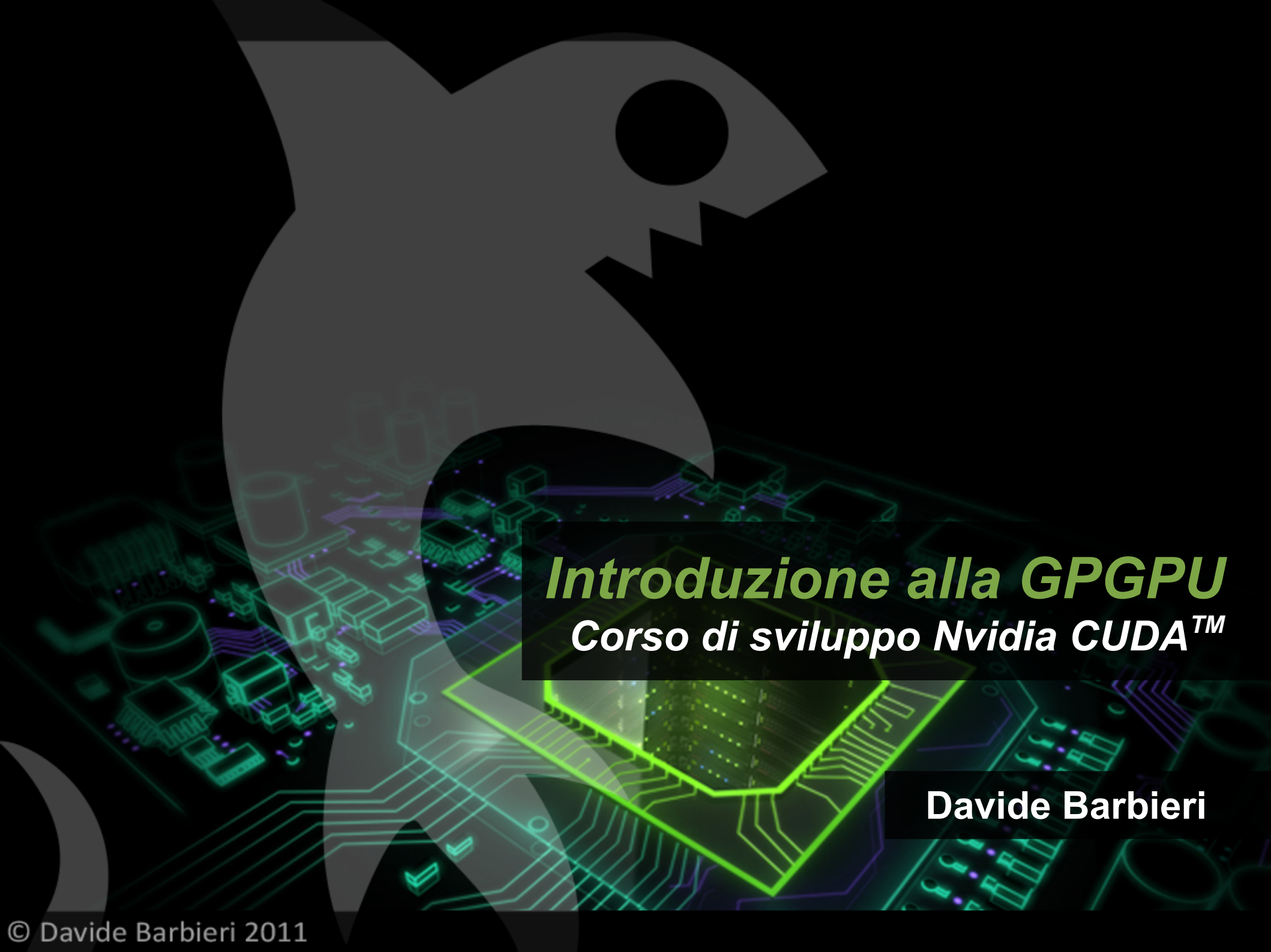


Introduzione alla GPGPU

Corso di sviluppo Nvidia CUDA™

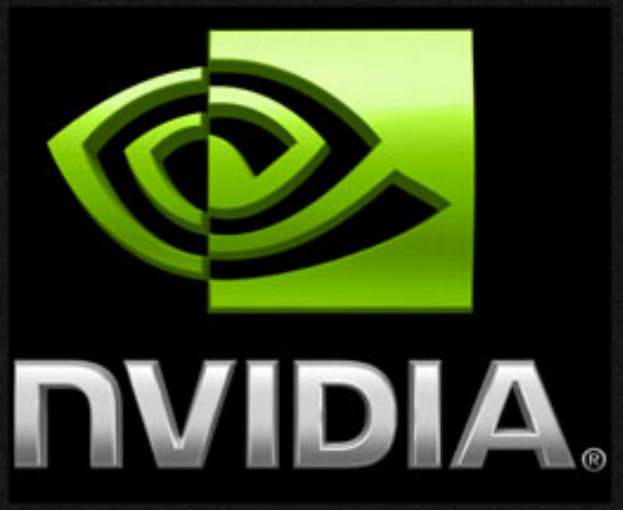
Davide Barbieri

Contatti

- skype: ***davbar86***
- mail: ***davide.barbieri@ghostshark.it***

Panoramica corso

- Introduzione al mondo delle **GPU**
- Modello **GPGPU**
- **Nvidia CUDA**
  - Architettura e modello di programmazione
  - Estensioni del linguaggio C/C++
  - Ottimizzazione delle prestazioni
- Accenni a soluzioni **multi-GPU**
- Sessioni di **laboratorio**



Panoramica Lezione

- Cos'è la GPU?
- Come è evoluta nel tempo?
- In cosa differisce dalla CPU?
- Cos'è la GPGPU?
- Cos'è CUDA?

Cos'è la GPU? (1/2)

- **Graphics Processing Unit**
  - Calcolo e disegno della scena 3D di un videogioco in *tempo reale*

- La scena è l'insieme delle geometrie 3D e dei loro materiali che compongono la parte *visibile* del gioco

- *Un videogioco 3D* ha la necessità di stampare a schermo con una frequenza di almeno 25 fotogrammi (*frame*) al secondo una scena 3D animata

- A differenza di un film 3D, lo stato della scena in ogni momento dipende dagli input del giocatore
  → in ogni momento la posizione di ogni singolo vertice deve essere ricalcolata
  → calcolo delle luci, degli effetti, disegno delle texture
- La GPU fa questo in hardware

Cos'è la GPU? (2/2)

- Sigla coniata da **Nvidia Corporation** nel 1999 per il debutto della *Nvidia Geforce 256:*
  "La GPU è un processore su singolo chip con integrated transform, lighting, triangle setup/clipping e rendering engines, capace di processare un minimo di 10 milioni di poligoni al secondo"

- ***Perché è necessario eseguire ciò in hardware dedicato?***

- Ogni geometria viene disegnata in modo differente a seconda delle sue caratteristiche (*materiale*)
  - Calcoli sui vertici che compongono le geometrie
  - Calcoli sul colore di ogni pixel dei triangoli da disegnare

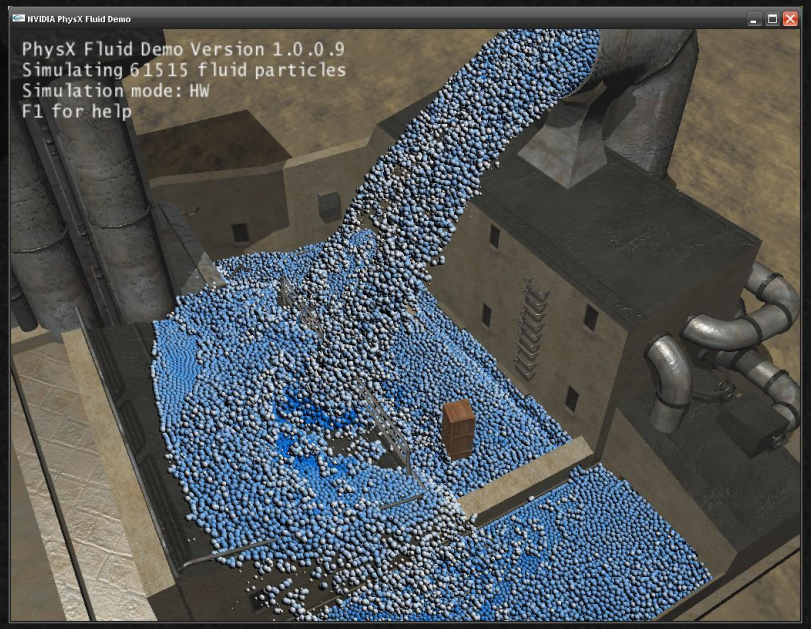
- Una scena può essere molto complessa
  - Centinaia di migliaia di triangoli
  - Milioni di pixel

- Ogni anno
  - Videogiochi graficamente sempre più complessi
  - Richiesta di prestazioni sempre maggiori

- Tanti calcoli, ma per lo più indipendenti tra loro
  → Alto parallelismo a livello di dati

Videogioco come simulazione

- Calcolo sulla geometria
  - Animazioni
- Risolutori di sistemi per il motore fisico del videogioco
  - Esempio PhysX 61000 particelle
- Intelligenza Artificiale
- Effetti di Rendering
- Altro



PhysX by NVIDIA

Evoluzione della scheda grafica (1/3)

- Nel 1999, la ***Nvidia Corporation*** inventa la prima GPU, **GeForce 256**
  - Integrava ***Transform&Lighting*** direttamente in hardware
    - Trasformazioni geometriche
    - Calcolo della luminosità dei pixel
  - Triangoli al secondo: 15 milioni
  - Pixel al secondo: 480 milioni

- ***ATI Technologies***
  - Acquisita da ***AMD***
  - Unica vera concorrente di Nvidia

- Pipeline fissa *(Fixed Function Pipeline)*
  - Funzionalità fisse
  - Configurazione delle funzionalità a *stati (Render State)*
  - Spesso lo stesso effetto aveva rese diverse da scheda a scheda

Evoluzione della scheda grafica (2/3)

- **Pipeline programmabile**
- Shader eseguiti
  - per ogni vertice (*Vertex Shader*)
  - per ogni primitiva (*Geometry Shader*)
  - per ogni pixel da colorare (*Pixel Shader*)

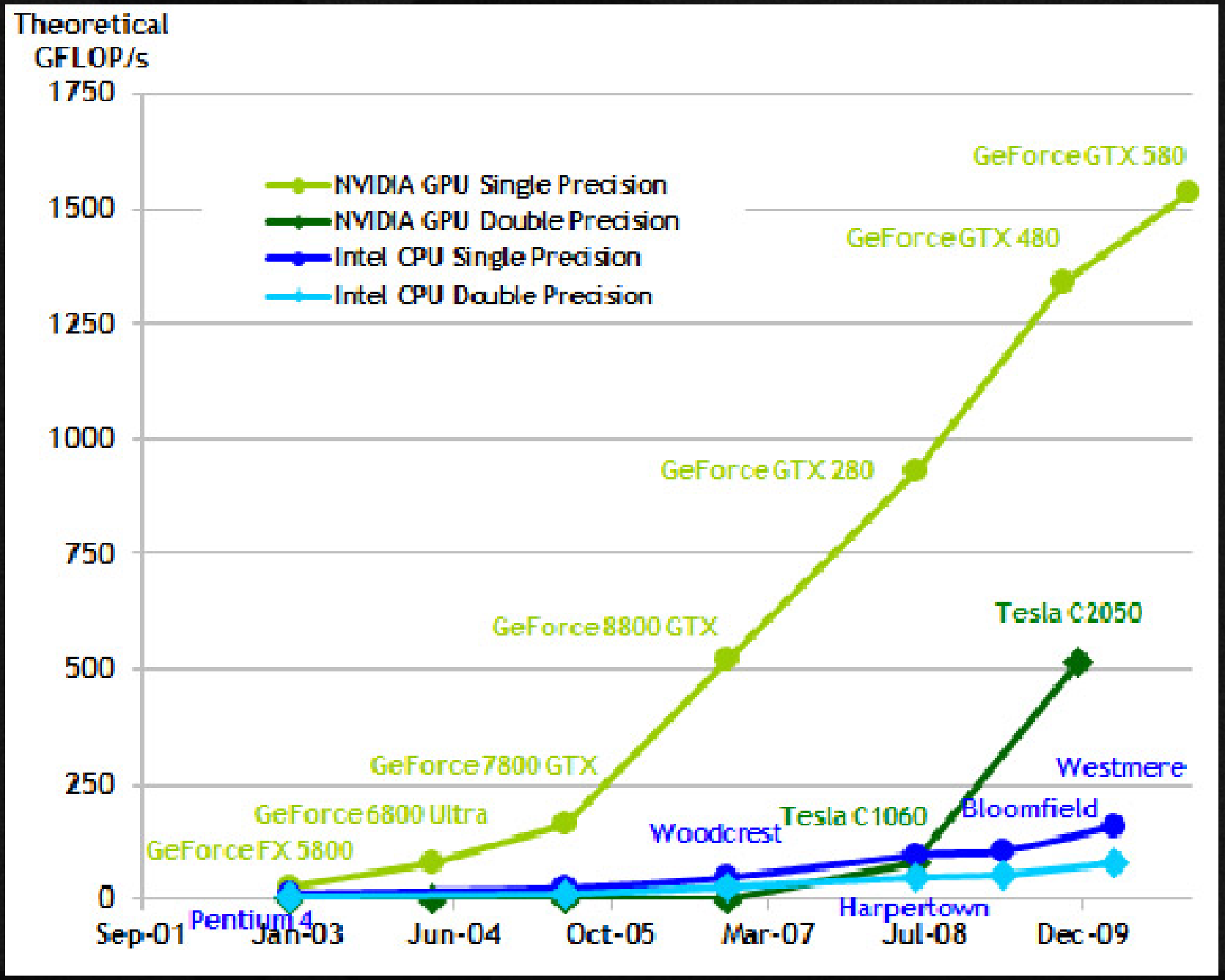
- Inizialmente semplici codici *Assembly*
  - Poi sempre più complessi

- Nascita di linguaggi simili al C
  - Nvidia **Cg**
  - Microsoft **HLSL**
  - **GLSL**

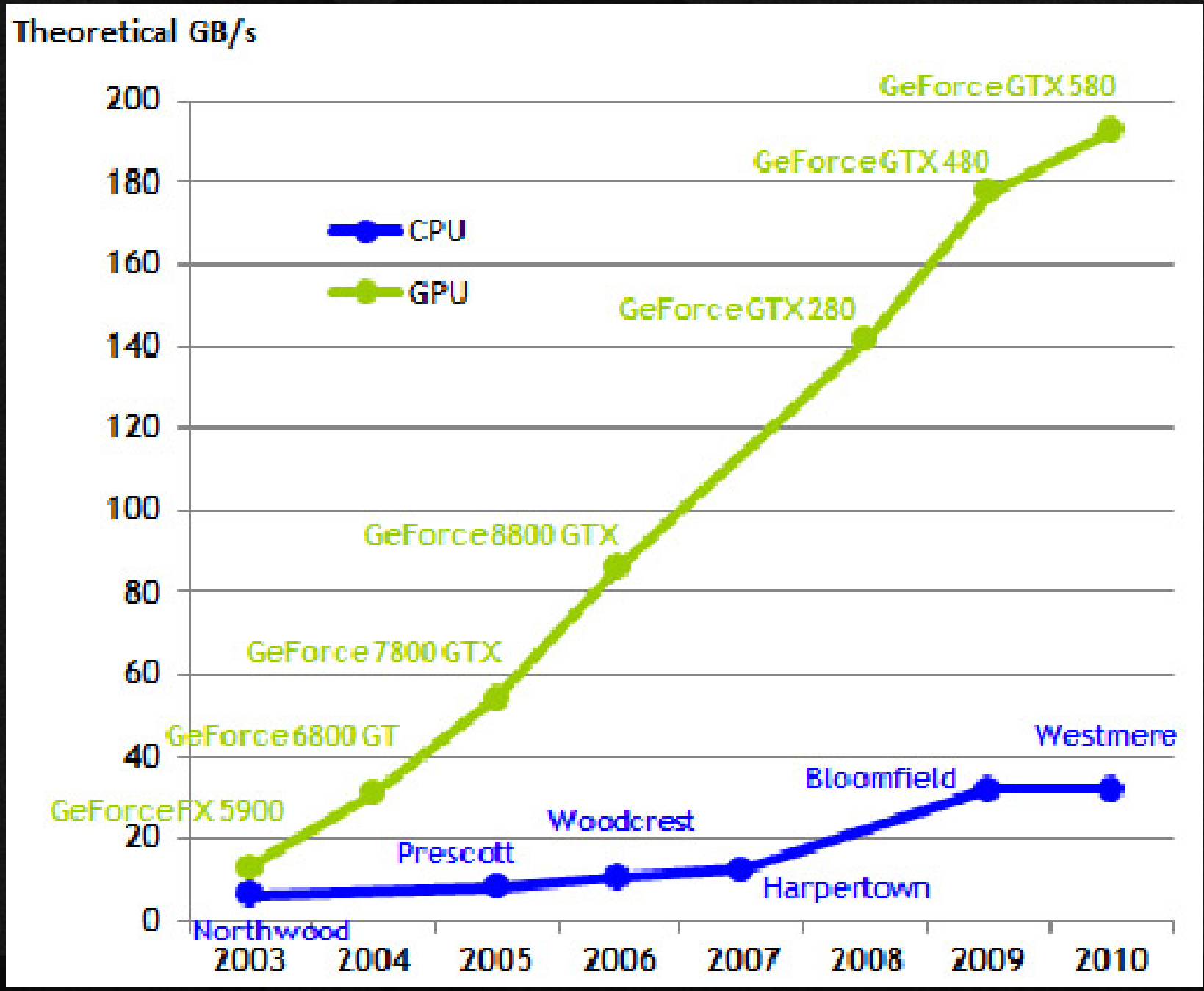
Evoluzione della scheda grafica (3/3)

- Maturità dello *shader model*
  - Controllo del flusso
  - Esecuzione predicata
  - *Unified Shader Model*
    - Set di istruzioni consistente per i diversi shader
- Architettura unificata
  - Vertex e Pixel shader eseguiti dagli stessi *core* *(Unified Shading Architecture)*
- Ha permesso la nascita di CUDA
  - **GPGPU: General Purpose computing using GPU**
    - Sviluppo di algoritmi di interesse generale
    - Senza necessità di programmazione grafica

Crescita prestazioni GPU (Operazioni ALU)



Crescita prestazioni GPU (Banda Memoria)



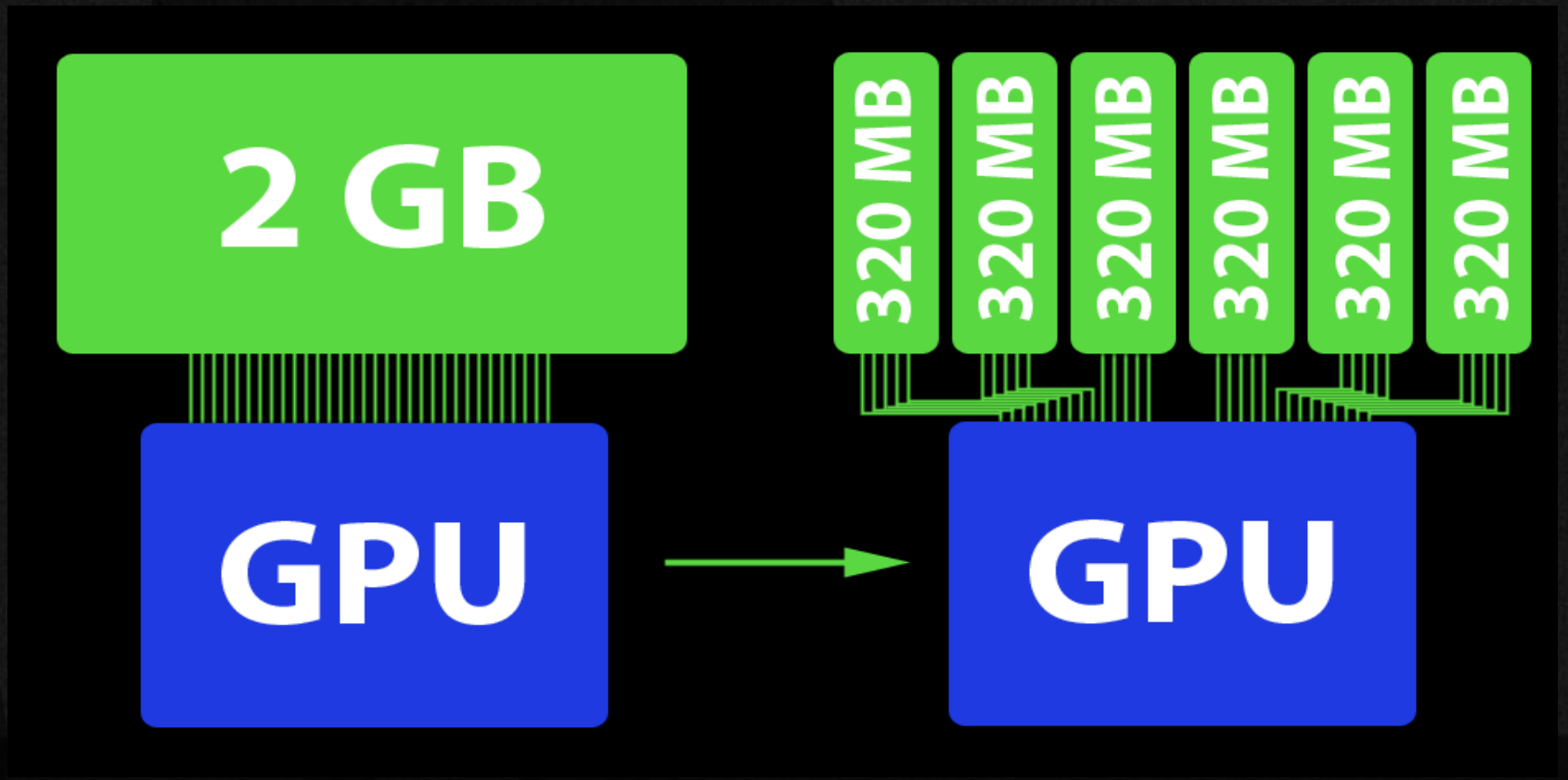
Alcune differenze con la CPU (1/3)

- **CPU**
  - Nasconde le latenze agli accessi in memoria tramite grandi (e ingombranti) ***cache***
    - ***e.g. AMD Phenom II***
      - ***6 cache L1 128KB***
      - ***6 cache L2 512 KB***
      - ***1 cache L3 6MB***
  - **Pochi core *(~4-6)*, pochi thread *(~6-8)* attivi**
    (che non necessitano context switching)
- **GPU**
  - Nasconde le latenze agli accessi in memoria tramite concorrenza di calcoli di migliaia di thread
  - Cache più piccole
    - e.g. Nvidia Geforce GTX 580
      - 16 memorie on-chip da 64 KB (fino a 48 KB per la Cache L1)
      - 768KB Cache L2
  - ***Tanti core, più semplici (~512), tanti thread (~16000) attivi***
    - ***Alto throughput di operazioni ALU***
  - ***Alta banda di memoria* (fino a 192 GB/s per GPU)**
    - ***rispetto ai 21 GB/s di CPU Intel Core I7-2600K***

Alcune differenze con la CPU (2/3)

- Perché la RAM GPU ha una banda così grande?
  - Bus molto grande (e.g. 384 bit per Geforce GTX 580)
  - (384/8 Byte) * 2004 Mhz * 2 (DDR) = ~192 GB/s

- In realtà la memoria è divisa in partizioni
  - Ad ogni partizione sono connessi solo 64 bit del bus
  - è come avere 6 memorie con banda 32 GB/s



Alcune differenze con la CPU (3/3)

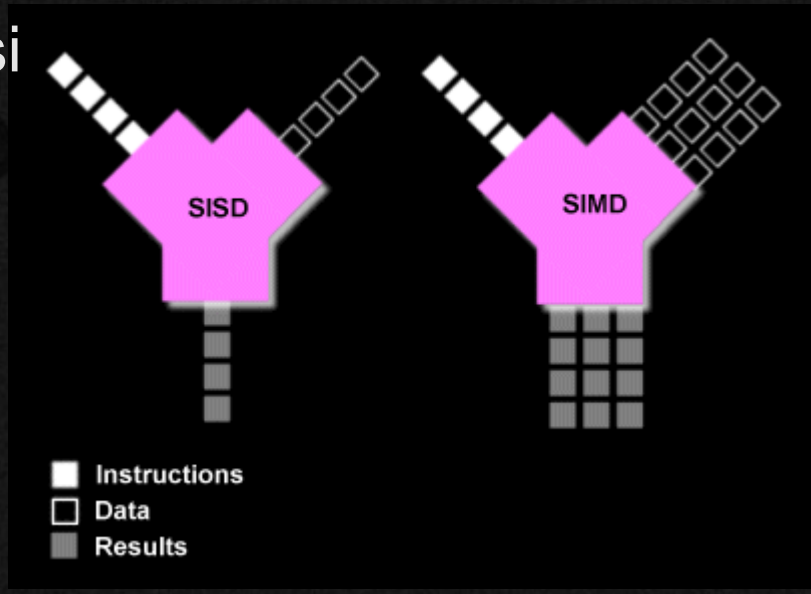
- Ad ogni partizione è associato un banco da 256 Byte
  - **0x0 ... 0xff** → Banco 0
  - **0x100 ... 0x1ff** → Banco 1
  - **0x200 … 0x2ff** → Banco 2

    ...
  - **0x500 … 0x5ff** → Banco 5
  - **0x600 … 0x6ff** → Banco 0
  - **0x700 … 0x7ff** → Banco 1

    …
- Per ottenere 192 GB/s dobbiamo accedere a tutti i banchi in parallelo
  - Se i multiprocessori della GPU, infatti, vogliono accedere allo stesso banco dovranno accedervi in sequenza
    - **Partition Camping**

SIMD vs SIMT

- ***SIMD*** – *Single Instruction Multiple Data*
  - Stessa istruzione eseguita su dati diversi
  - Esempio:
    - Per 256 volte
      - Carica 4 float nell'array A
      - Carica 4 float nell'array B
      - Moltiplica le 4 componenti dei due array
        C(i) = A(i) * B(i)
  - e.g. Estensioni *SSE** degli x86/x86-64



- ***SIMT*** – *Single Instruction Multiple Threads*
  - Stessa istruzione eseguita da thread diversi
    (possibilmente su dati diversi)
  - Esempio:
    - Esegui C(***i***) = A(***i***) * B(***i***) su 1000 thread con id ***0≤i<1000***
  - Può sembrare uno spreco
    - Thread "*leggeri*"
    - Scheduling con **Zero Overhead**
      - Risorse disponibili da dispatching a fine esecuzione
      - **Nessun Context-Switching**

Dove la GPU "perde"?

- Picchi prestazionali più alti per la GPU
  - Raggiungibili solo per problemi con **Parallelismo a livello di dati**
    - **Stessa operazione, più dati**

- La CPU sarà sempre più veloce per algoritmi **intrinsecamente sequenziali**

- Spesso algoritmi CPU devono essere stravolti
  - Limitare al massimo la dipendenza delle soluzioni ad ogni passo
  - Spesso, per farlo, conviene ripetere gli stessi calcoli
  - Programmazione dinamica difficile da implementare
    - e.g. Algoritmi ricorsivi

- Ottimizzazione delle prestazioni è spesso indispensabile
  - Piccole differenze nel codice possono implicare ordini di grandezza di differenza nelle prestazioni

Cos'è CUDA?

- **Compute Unified Device Architecture**

- Architettura per la GPGPU

- Linguaggio CUDA C/C++
  - Basato su C/C++
  - Estensioni per la programmazione parallela
  - API per la gestione delle risorse GPU

- Ambiente di sviluppo
  - Toolkit (Compilatore e altri tool)
  - SDK
  - Librerie
    - CUBLAS – Calcolo con matrici dense
    - CUSPARSE – Calcolo con matrici sparse
    - CUFFT – Fast Fourier Transform
    - CURAND – Generatori pseudocasuali su GPU