Nvidia CUDA

Organisation générale du module

- 2 CMs (9/03 et 16/03) – 1h30
- 6 TPs (Vendredi Matin) – session de 2h
- Evaluation : projet

Agenda

1. Introduction et Enjeux
2. Architecture
3. Modèle de programmation
4. Modèle d’exécution
5. C/C++ API
6. Bilan

1 – NVIDIA CUDA

Compute Unified Device Architecture

1. Cartes graphiques Nvidia
2. + pilotes CUDA
3. + extensions de langage C/C++
4. + compilateurs, outils, libraries.

1 – Environnement Logiciel

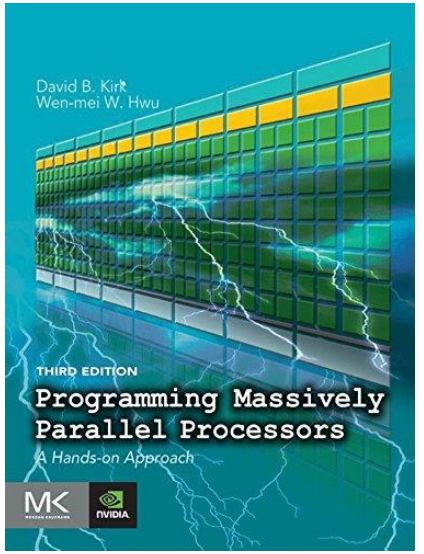
Téléchargement sur le site

https://developer.nvidia.com/cuda-downloads

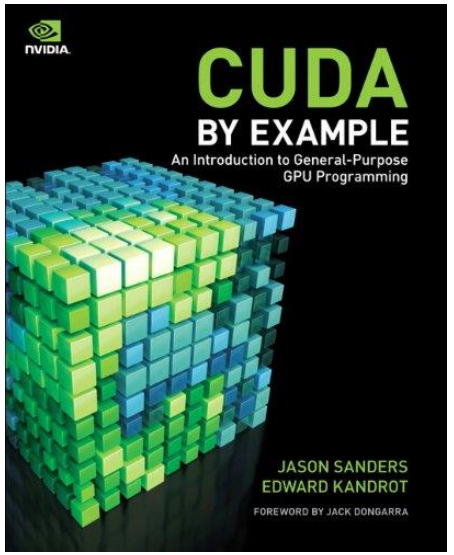
1. Pilotes Nvidia
2. CUDA Toolkit : compilateur, debugger, documentation, exemples
3. Nvidia Nsight (Visual Studio ou Eclipe)
4. Bibliothèques : CuBLAS, CuFFT …

1 - Ressources

- David B.Kirk et Wen-mei W. Hwu : *Programming Massively Parallel Processors*



- Jason Sanders et Esward Kandrot : *CUDA by Example*



1 - Ressources

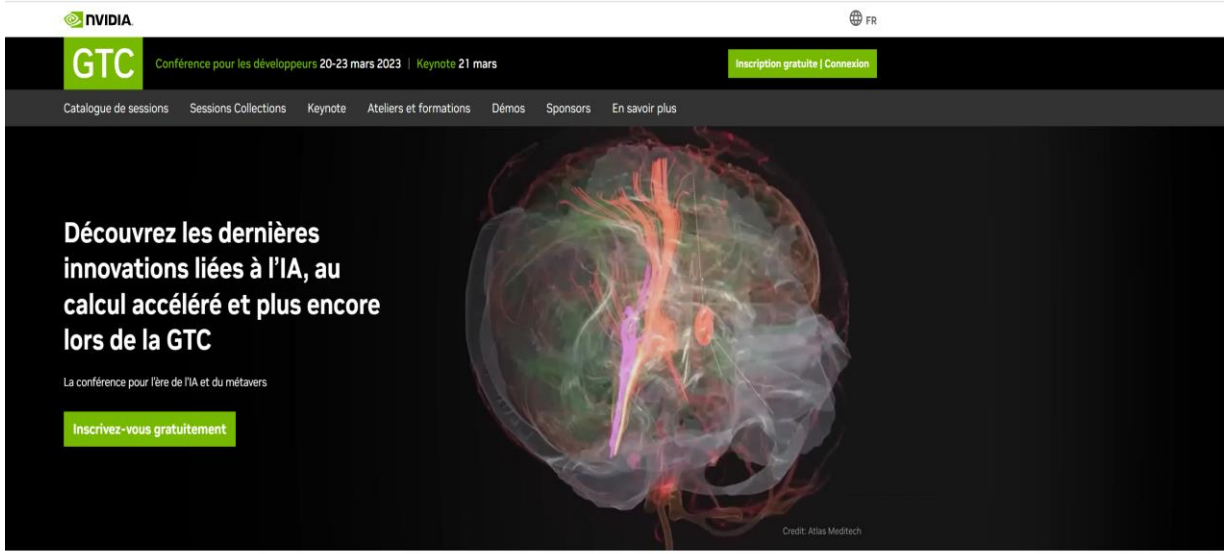
- Nvidia Developer et CUDA zone

✓ *https://developer.nvidia.com/cuda-zone*

- Conférence annuelle GTC

✓ *https://www.nvidia.com/fr-fr/gtc/*

✓ *20-23 Mars 2023 (keynote 21 Mars)*



Rejoignez-nous en ligne du 20 au 23 mars

Venez découvrir un contenu encore plus inspirant à NVIDIA GTC, des conférences animées par des experts et un discours d'ouverture incontournable pour accélérer le travail de notre vie.

1 – Domaines applicatifs

- 3D, IA, Simulations, Metaverse ….

1 – Bilan

- Nvidia CUDA : plus large qu’une API pour les GPUs
- Environnement complet : outils, librairies …
- Programmation simple, optimisation itérative

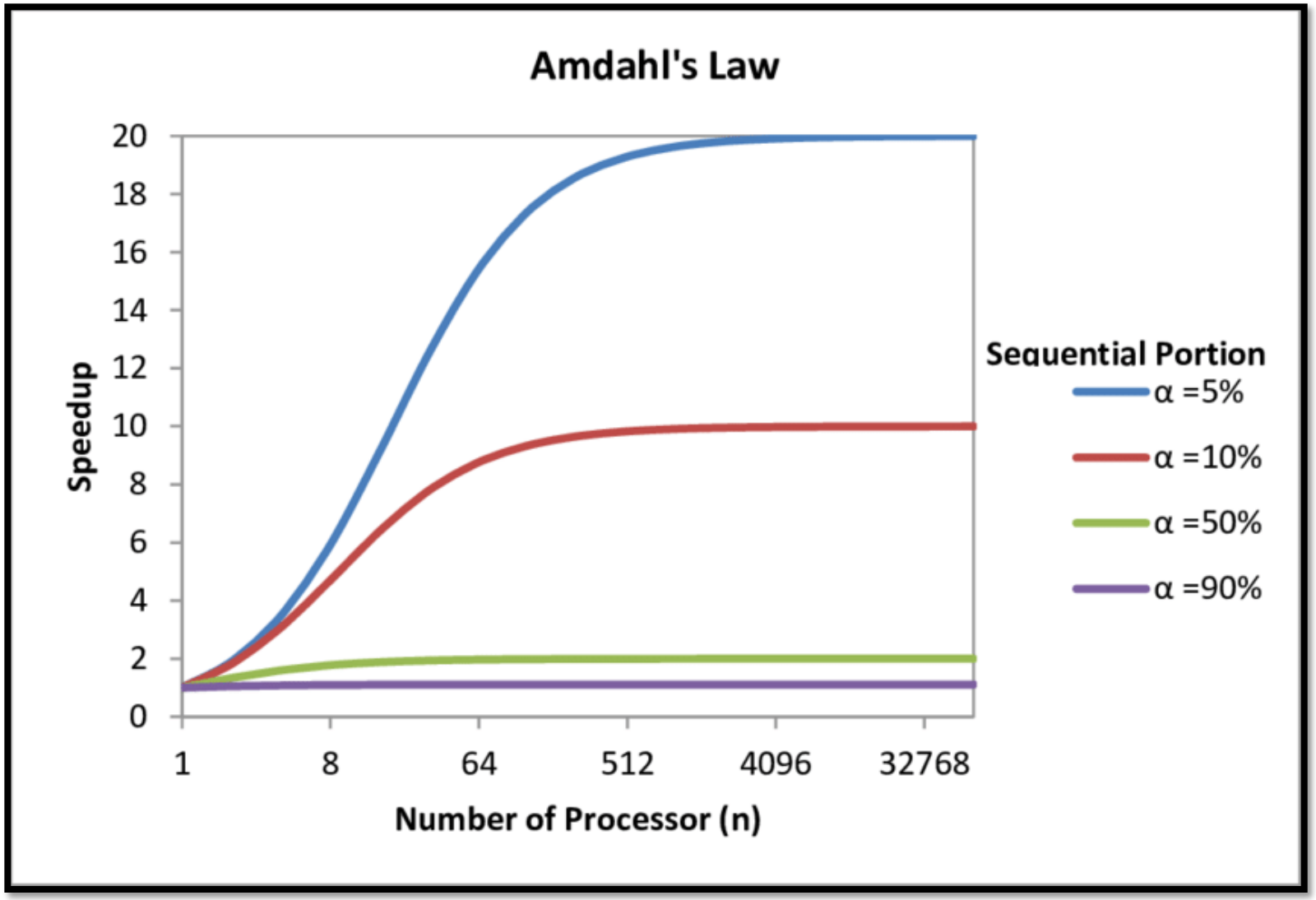
Agenda

1. Introduction et Enjeux
2. Architecture
3. Modèle d’exécution

2 – Taxonomie de Flynn

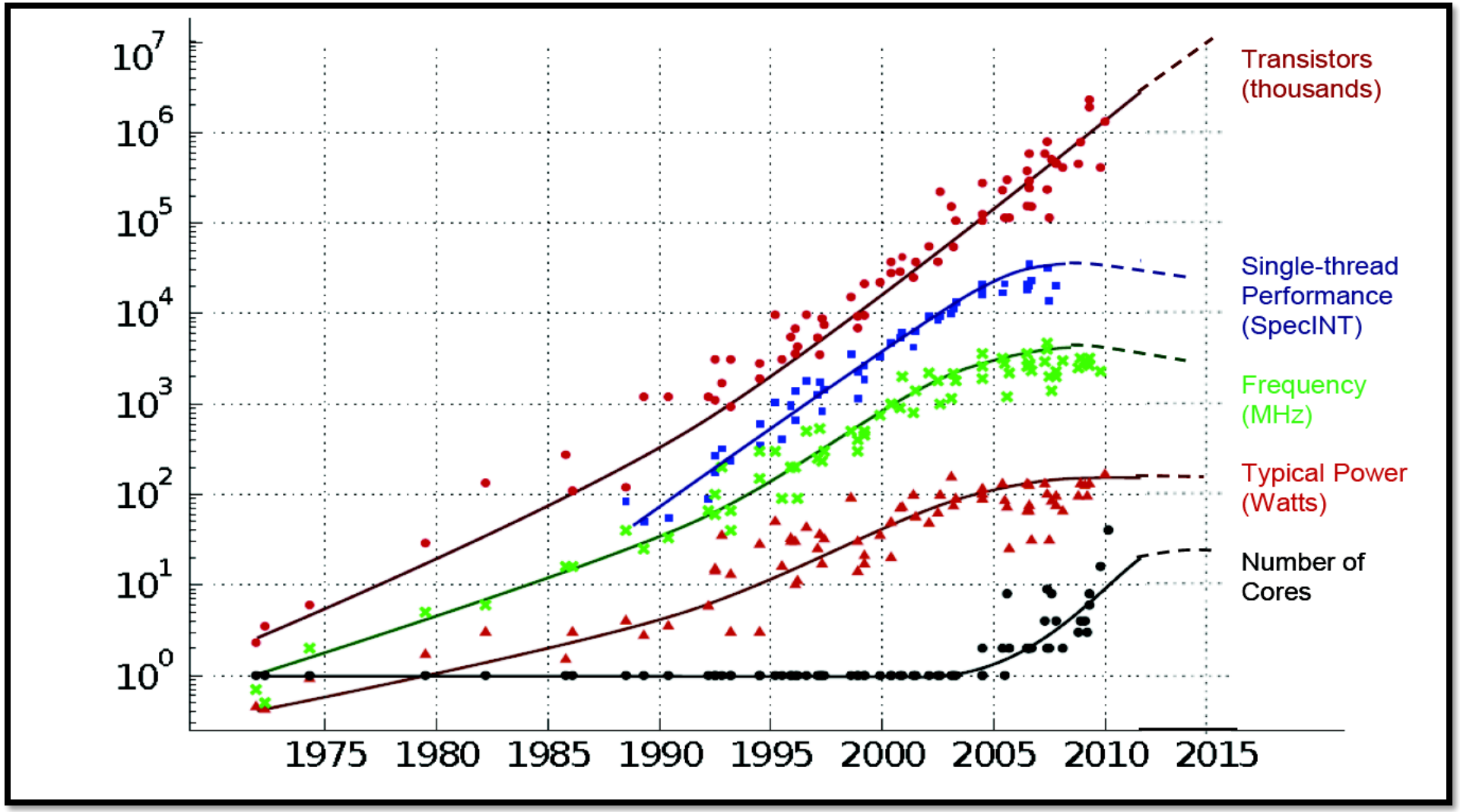
	Single Data	Multiple Data
Single Instruction	**SISD**	**SIMD**
Multiple Instruction	**MISD**	**MIMD**

2 – Loi Amdhal

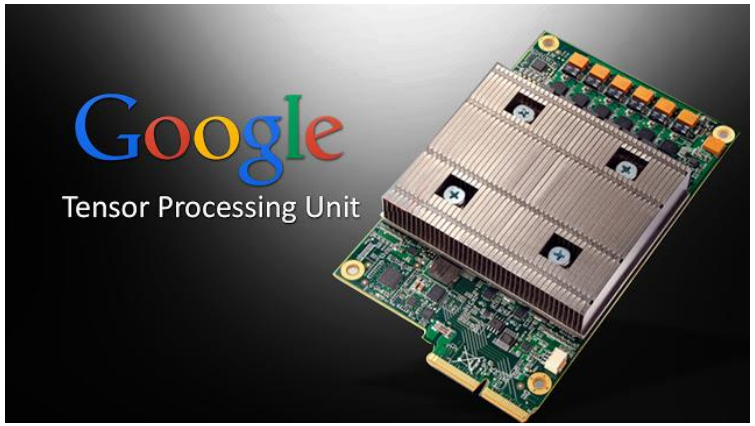


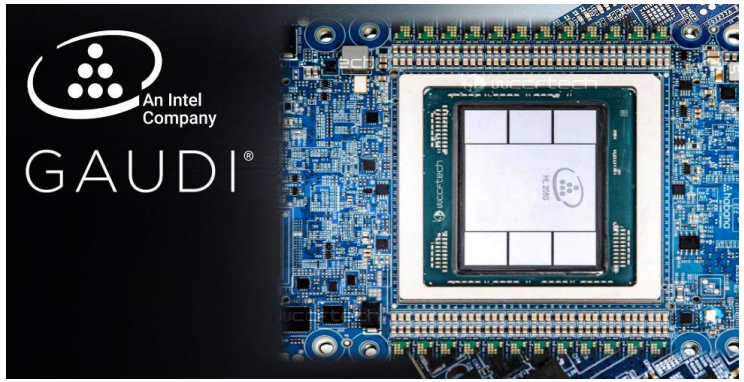
$$Speedup = \frac{1}{(1-p) + p/N}$$

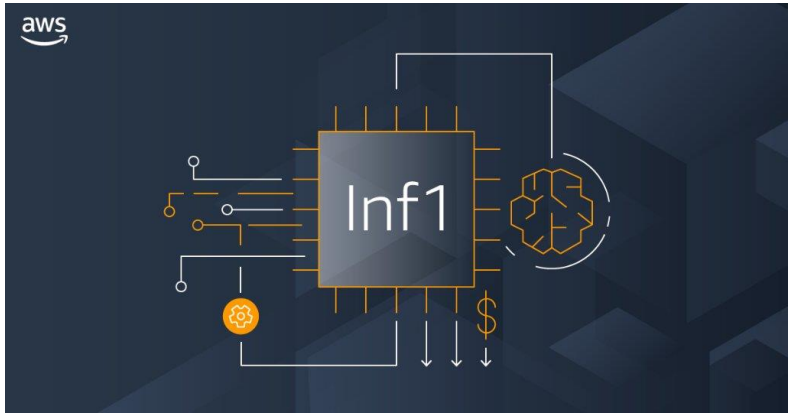
2 – Loi de Moore



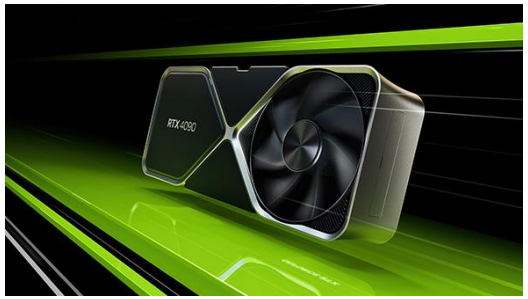
2 – Pourquoi les accélérateurs ?

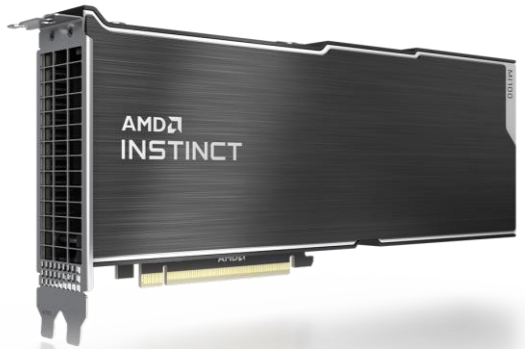




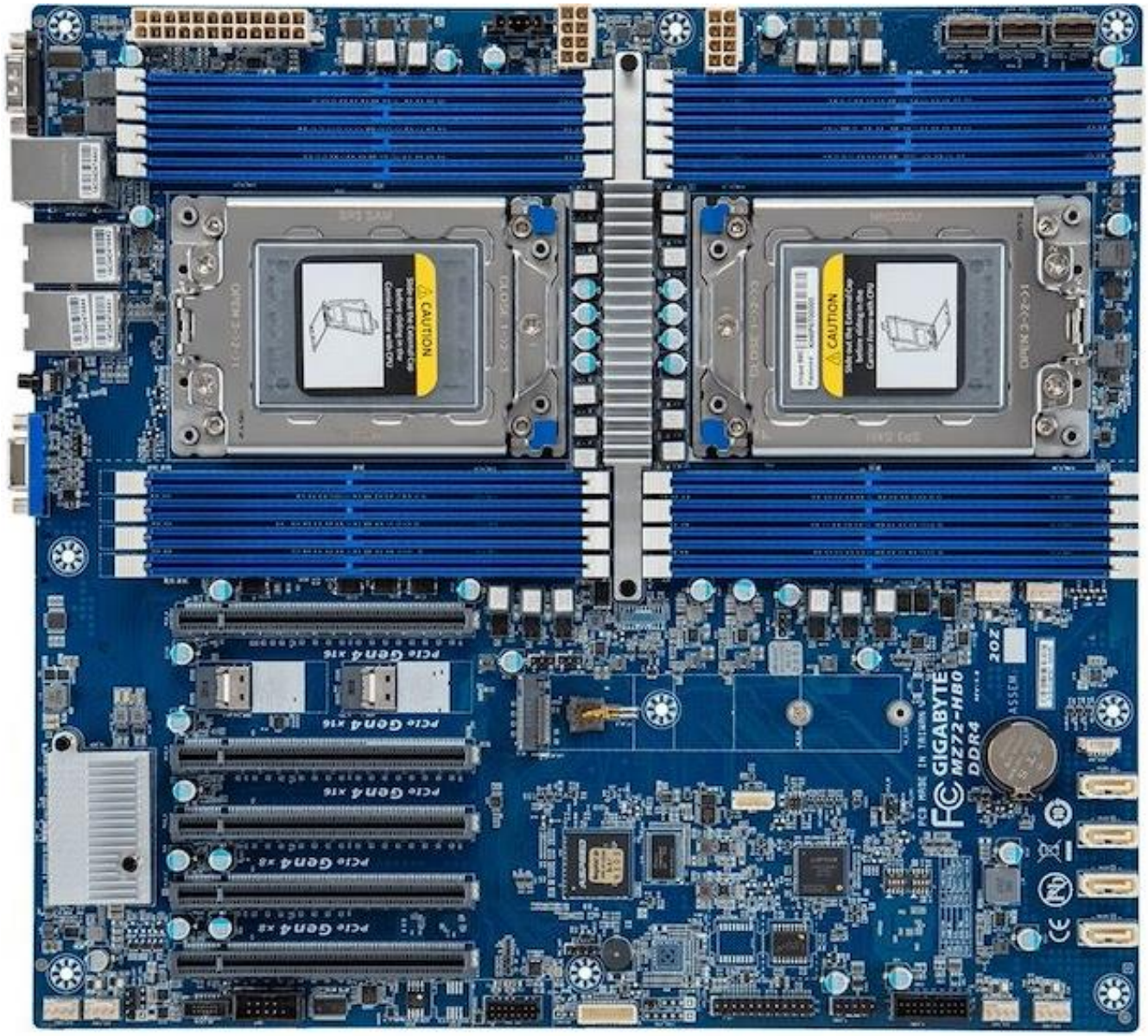


2 – Pourquoi les GPUs ?

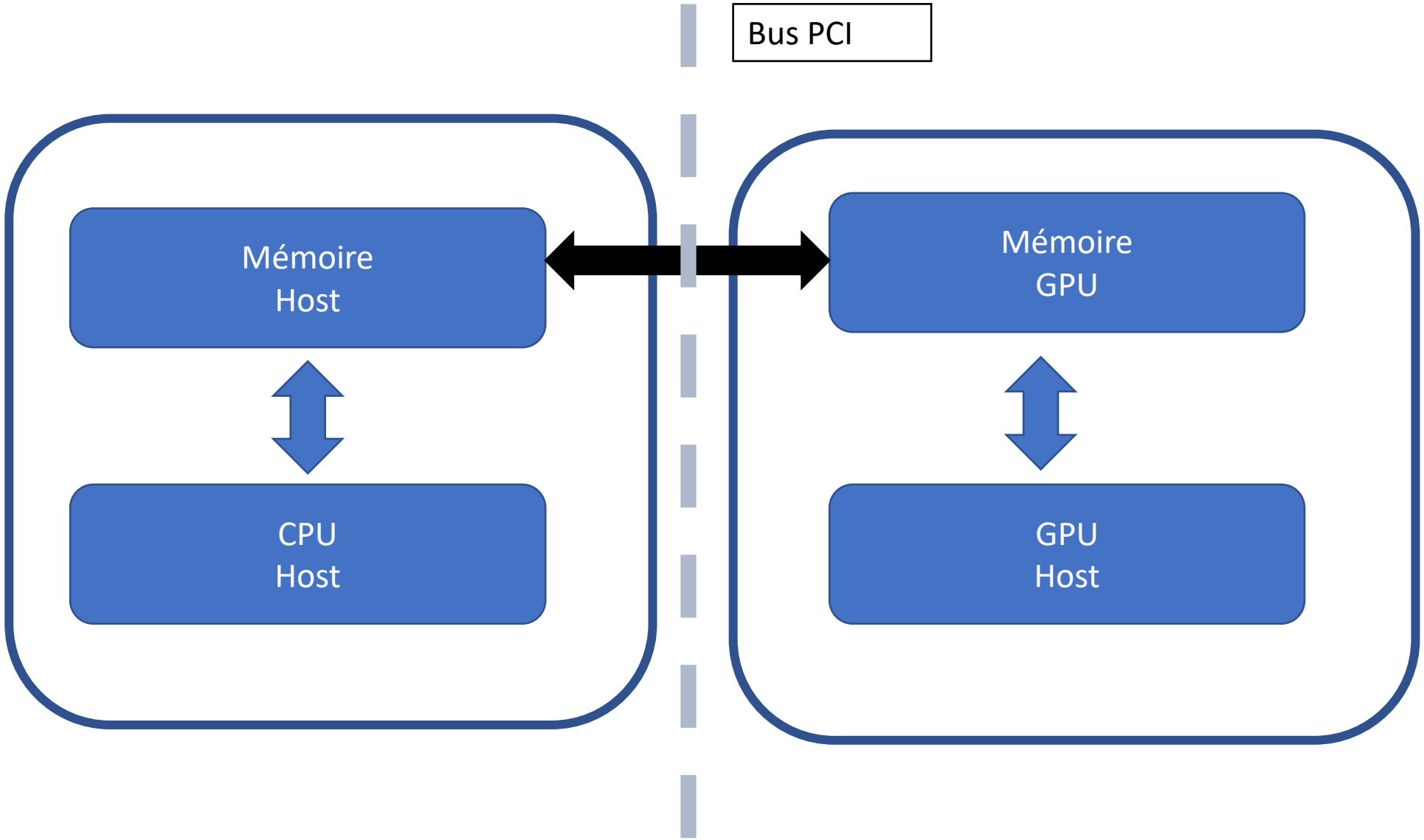




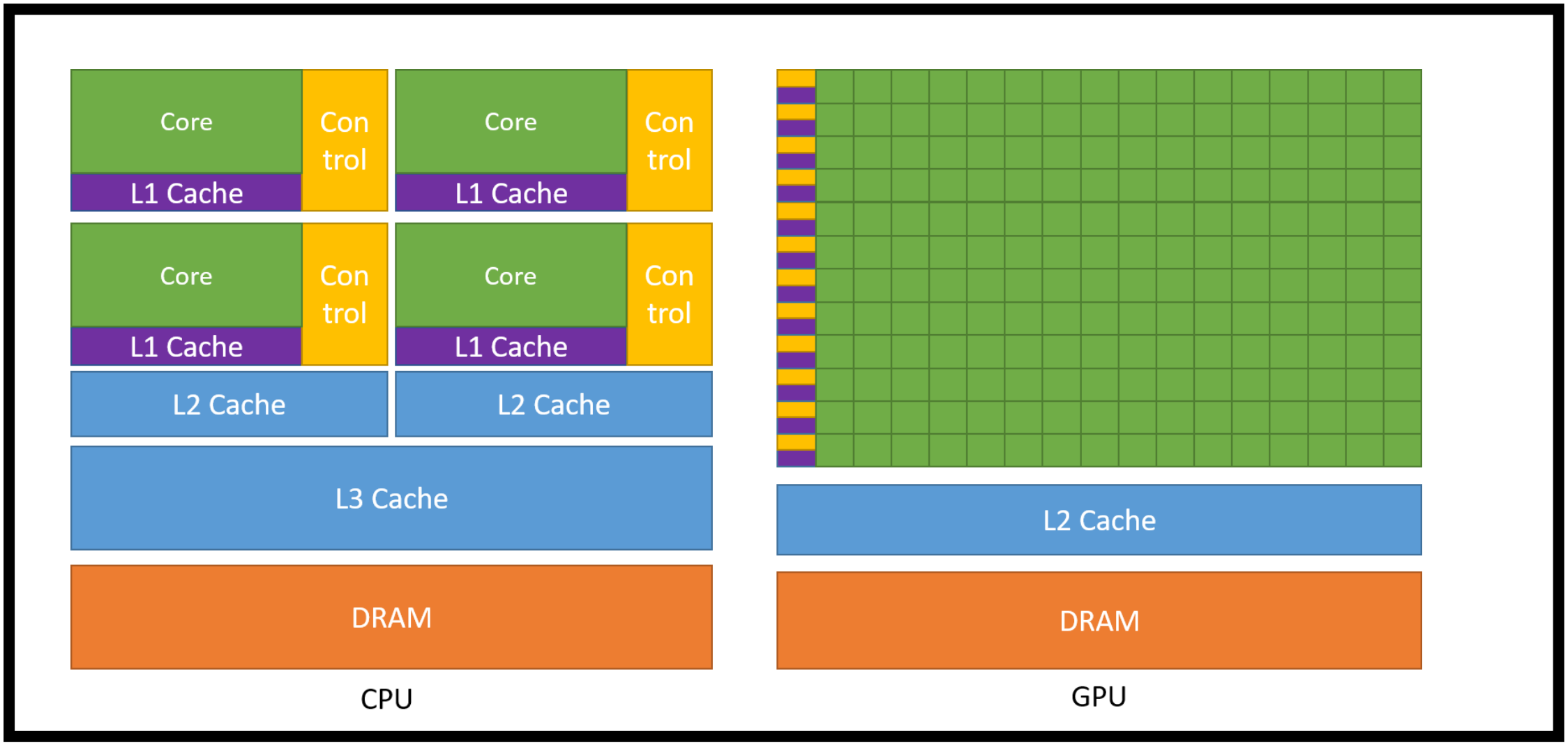
2 – Architecture générale



2 – Architecture générale



2 – Architecture générale



2 – Gammes Nvidia

- GeForce : gamme grand public (jeu)
- Quadro : gamme professionnelle pour la 3D
- Tesla : gamme professionnelle pour GPGPU : pas de composants video, mémoire ECC.
- Tegra : gamme pour les plateformes nomade (tablettes, smartphones, voitures autonomes)

2 – Générations de GPU

Générations	Année
Tesla *Cuda*	*2008*
Fermi	*2010*
Kepler *GPU Boost, GPUDirect, Dynamic Parallelism*	*2012*
Maxwell	*2014*
Pascal *HBM2, NVlink*	*2016*
Volta *Tensor Cores*	*2017*
Turing	*2018*
Ampere	*2020*

Compute Capabilities

https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

2 – Mémoires

Dans le GPU : mémoires rapides mais de taille limitée

- Registres
- Mémoire partagée, gérée par le développeur
- Caches : mémoire de type constante et texture

Dans la DRAM : mémoire plus lente mais en grande quantité

- Mémoire locale et globale
- Mémoire de type constante et texture

2 – Mémoires

Type	On-chip	Cached	Accès	performance
• Registres • Shared	oui oui	- -	rw rw	1 cycle 1 cycle
• Local • Global • Constant • Texture	non non non non	non non oui oui	rw rw r r	~100 cycles ~100 cycles 1 .. 10 .. 100 1 .. 10 .. 100

2 – Unités de calcul

- Cuda core - Streaming Processor
  - ✓ Unité Arithmétique
  - ✓ Registres

- Cuda Streamings Multiprocessors (SMs)
  - ✓ Attaché à la mémoire globale
  - ✓ Plusieurs Cuda Core

- Cuda GPU
  - ✓ Plusieurs Streaming Multiprocessors (SMs)
  - ✓ Mémoire globale

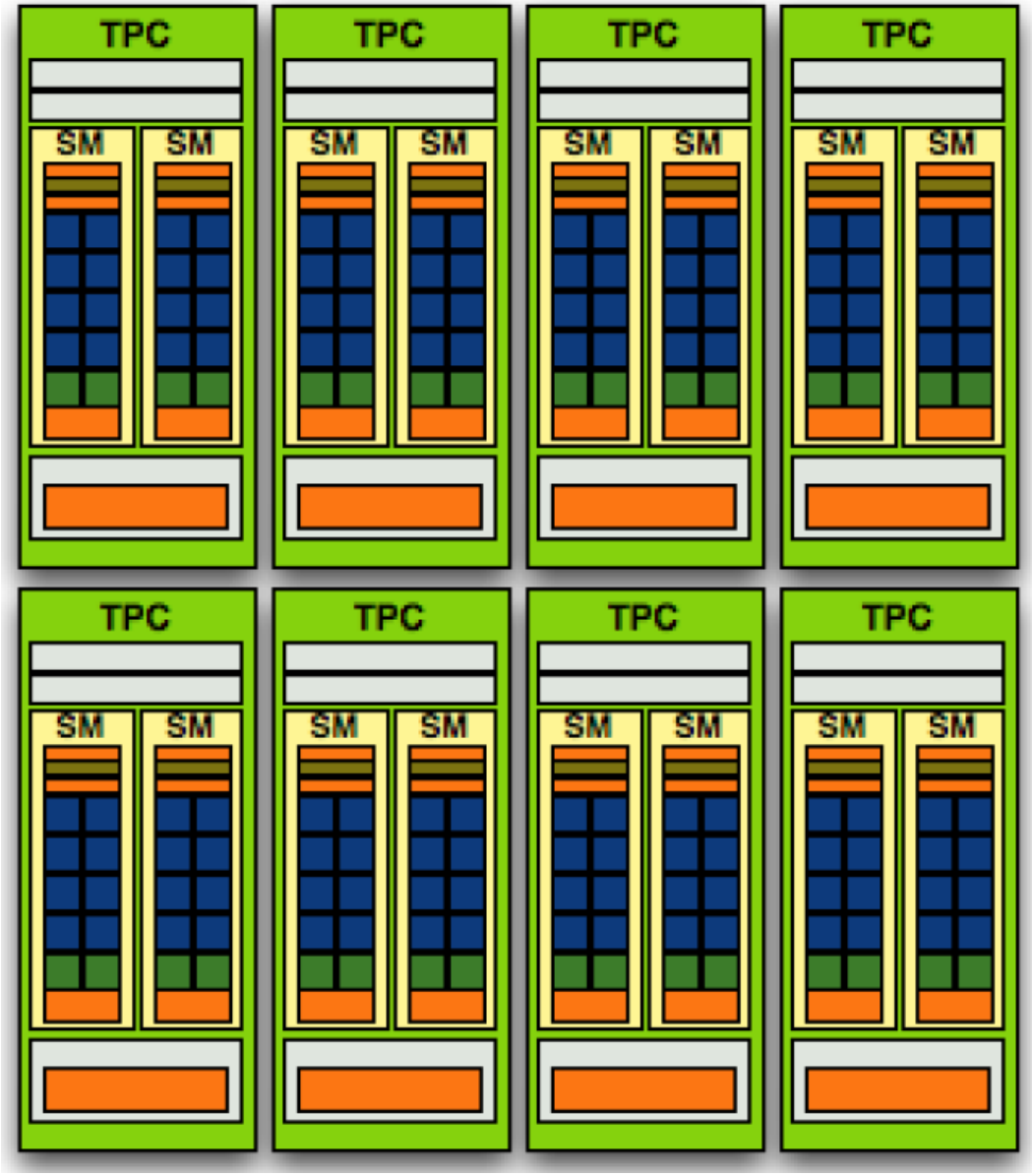
2 – Architecture Tesla (G80)

- Unités de calcul regroupées par 8

→Multi-Processeurs (SMs)

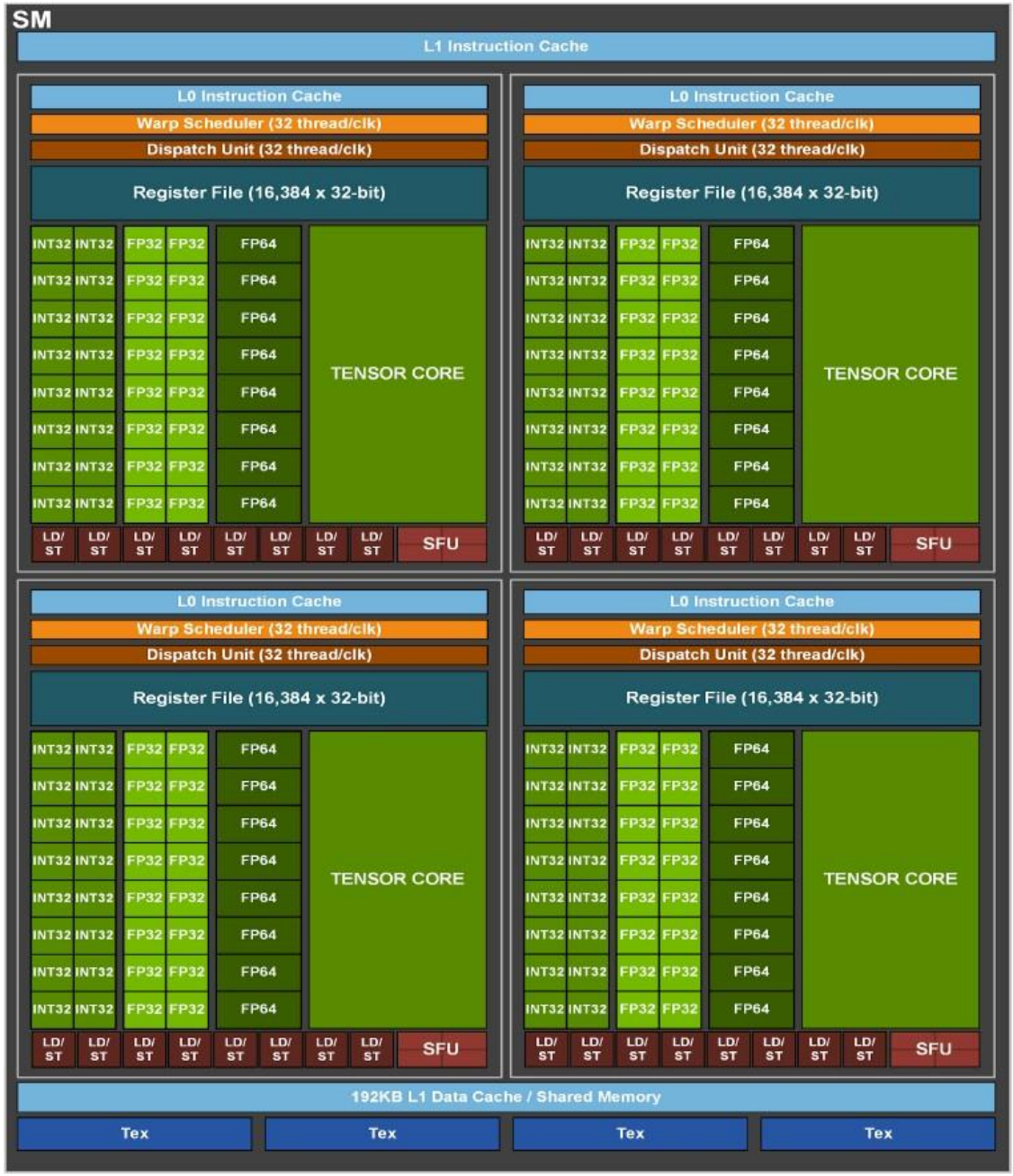
- Multi-Processeurs regroupés par 2 (G80)

→ Texture Processing Clusters (TPC)

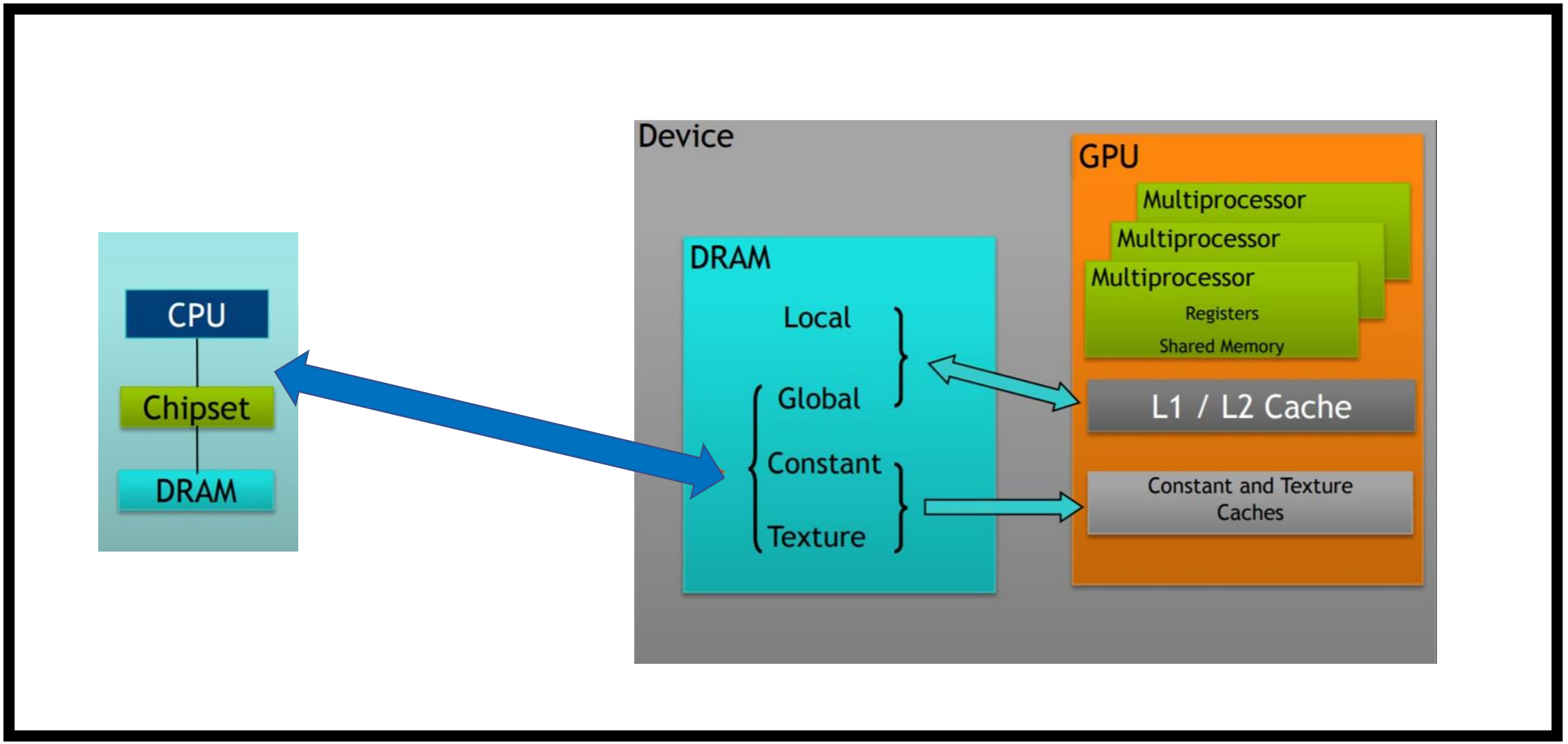


2 – Architecture Volta (V100)

- Unités de calcul entier, simple et double precision
- Tensor cores
- 80 SMs , +5000 CUDA cores
- Mémoires : HBM2, L2 cache



2 – Mémoires / Unités de Calcul



2 – Architectures : comparatif

Modele Tesla	K80 Kepler	M60 Maxwell	P100 Pascal	V100 Volta	A100 Ampere
Bus	PCI	PCI	PCI	PCI	PCI
Fréquence (MHz)	560	899	1126	1372	1265
Streaming Multiprocessors	2x13	2x16	56	80	108
FP32 cores	4992	4096	3584	5120	6912
FP64 cores	-	-	1792	2560	3456
Tensor cores	-	-	320	432	432
Global Memory	24	16	16	32	40
FP32 (Teraflops)	8.74	9.6	10.6	14.0	19.5
TF32 (Teraflops)	-	-		125	312
Bande Passante (GB/s)	480	320	720	900	1555
Power (Watts)	300	250	250	250	400

2 – Bilan

- Plusieurs générations : Tesla, Fermi, Kepler …
- Optimisations spécifiques à chaque architecture.
- Connaissance approfondie des architectures et de leur fonctionnement.

Agenda

1. Introduction et Enjeux
2. Architecture
3. Modèle de programmation

3 – Programmation hétérogène

Un programme contient à la fois :
- Un code **host**, qui sera exécuté par le CPU,
- Le code **device** ou **kernel**, qui sera exécuté par le GPU.

Code host
- Fonctions CPU
- les communications entre la mémoire **host** et **device.**

Code device ou Kernel
- Un **kernel** est une procédure (pas de valeur retour)
- Exécution par les cœurs du GPU (portion parallèle de programme).
- Un kernel peut appeler un autre kernel

3 – Coté device : Modèle SPMD

Parallélisme de données

- Modèle SPMD : Single Program on Multiple Data
- A l'exécution, le code host va instancier un kernel

Thread

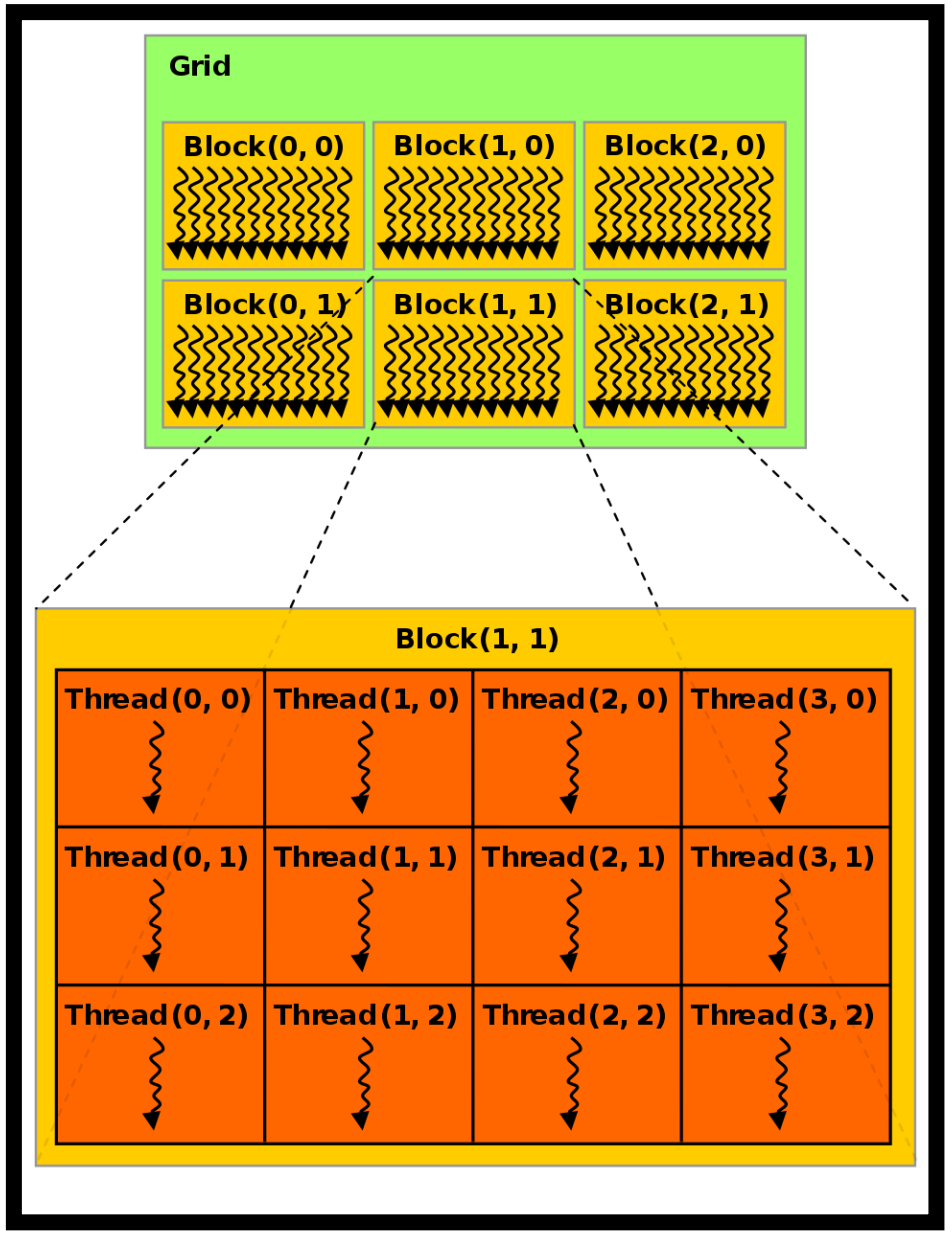
- Une instance de kernel est appelé thread → *modèle de threads*
- Grand nombre (plusieurs 100s/1000s) → *ressources matériel*
- Tous les threads d'un même kernel exécutent le même code mais peuvent prendre des chemins différents en cas de blocs conditionnels (e.g., threadId) → *efficacité…*
- Tous les threads ne s'exécutent pas en même temps → *ordonnancement/asynchronisme*

3 – Organisation des threads

- Les threads sont regroupés en blocs
- Les blocs sont groupés en une grille

⇒ *Exécuter un kernel : exécuter une grille de blocs de threads*

Chaque thread/bloc possède un identifiant (1D, 2D, 3D).



3 – Remarques

Choix du nombre de threads, de la topologie :

- Sur le CPU : nombre de threads correspondant au nombre de cœurs disponibles
- Sur le GPU : nombre de threads correspondant au nombre de données à traiter

Exemples:

- Somme de 2 vecteurs 1D de n flottants → n threads, topologie 1D
- Traitement d'une image de Nx.Ny pixels → blocs 2D *(tx, ty)* de threads

3 – Déroulement d'un programme CUDA

1. Initialisation des mémoires
- Initialisation des données en mémoire sur le host
- allocation dans la mémoire global device : cudaMalloc

2. Copie des données de la mémoire host vers la mémoire : device cudaMemcpy

3. Exécution du kernel sur les données en mémoire device lancement des threads sur le GPU
- copie des résultats en mémoire device vers la mémoire host cudaMemcpy
- exploitation directe des résultats par OpenGL pour affichage

4. Libération de la mémoire globale device cudaFree

3 – Compilation

Le code host et device peuvent-être dans le même fichier
L'extension du fichier est .cu.
Le compilateur nvcc sépare les codes **host** et **device**.

- Le code **host** est compilé par le compilateur C/C++ par défaut
- Le code **device** est compilé par le compilateur CUDA puis le binaire est inséré dans l'exécutable

Bilan Global

- Nvidia CUDA – multiples domaines applicatifs.
- Architecture massivement parallèle.
- Modèle de programmation / hiérarchie de l’architecture.

- Concepts clés de programmation demeurent abordables.

Fin