

# Introduction au calcul GP-GPU

## Partie 1 : Mise en contexte

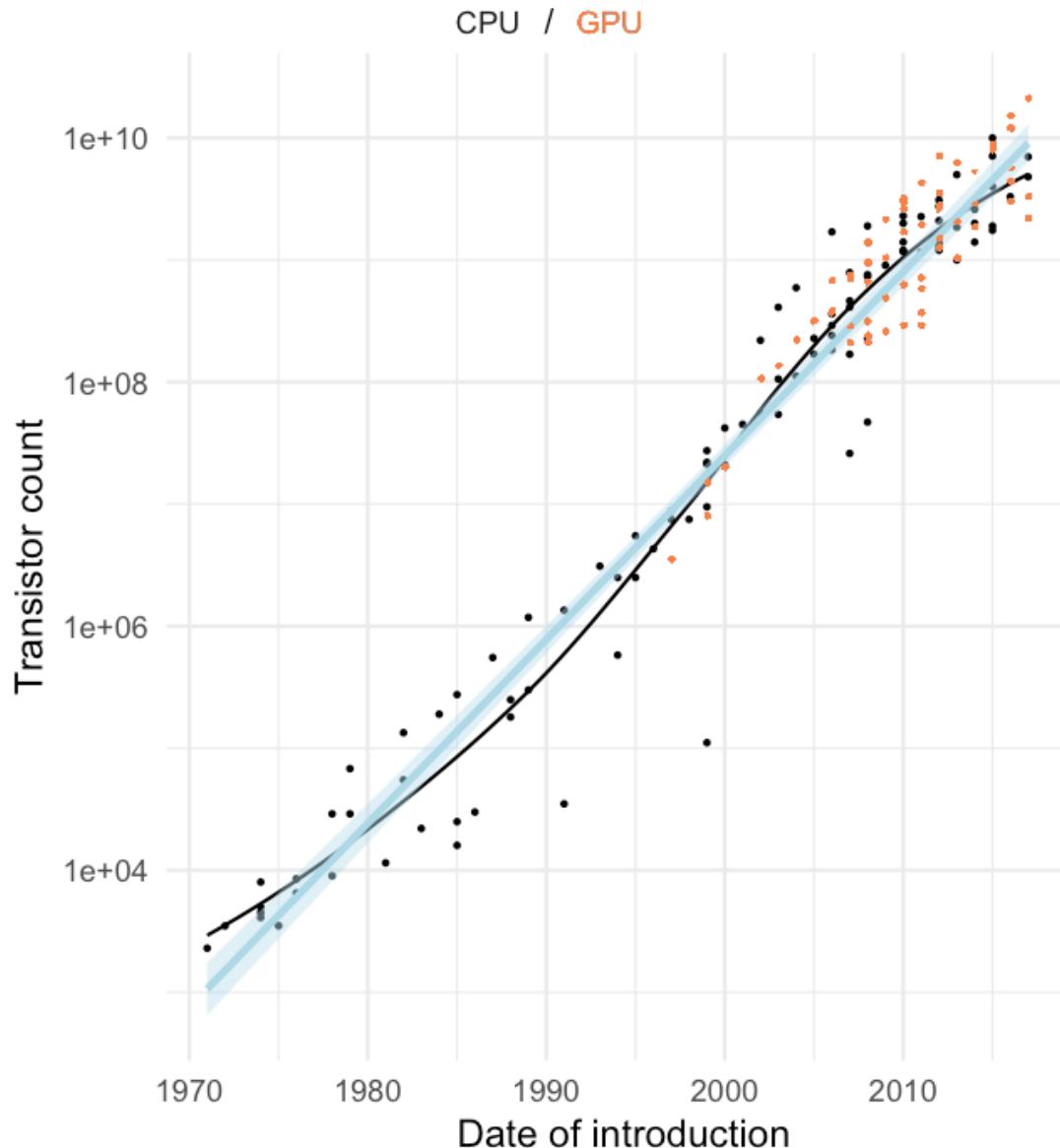
Laurent Risser

Institut de mathématiques de Toulouse  
[lrisser@math.univ-toulouse.fr](mailto:lrisser@math.univ-toulouse.fr)

# Préambule - loi de Moore

Moore's Law continued

Still linear



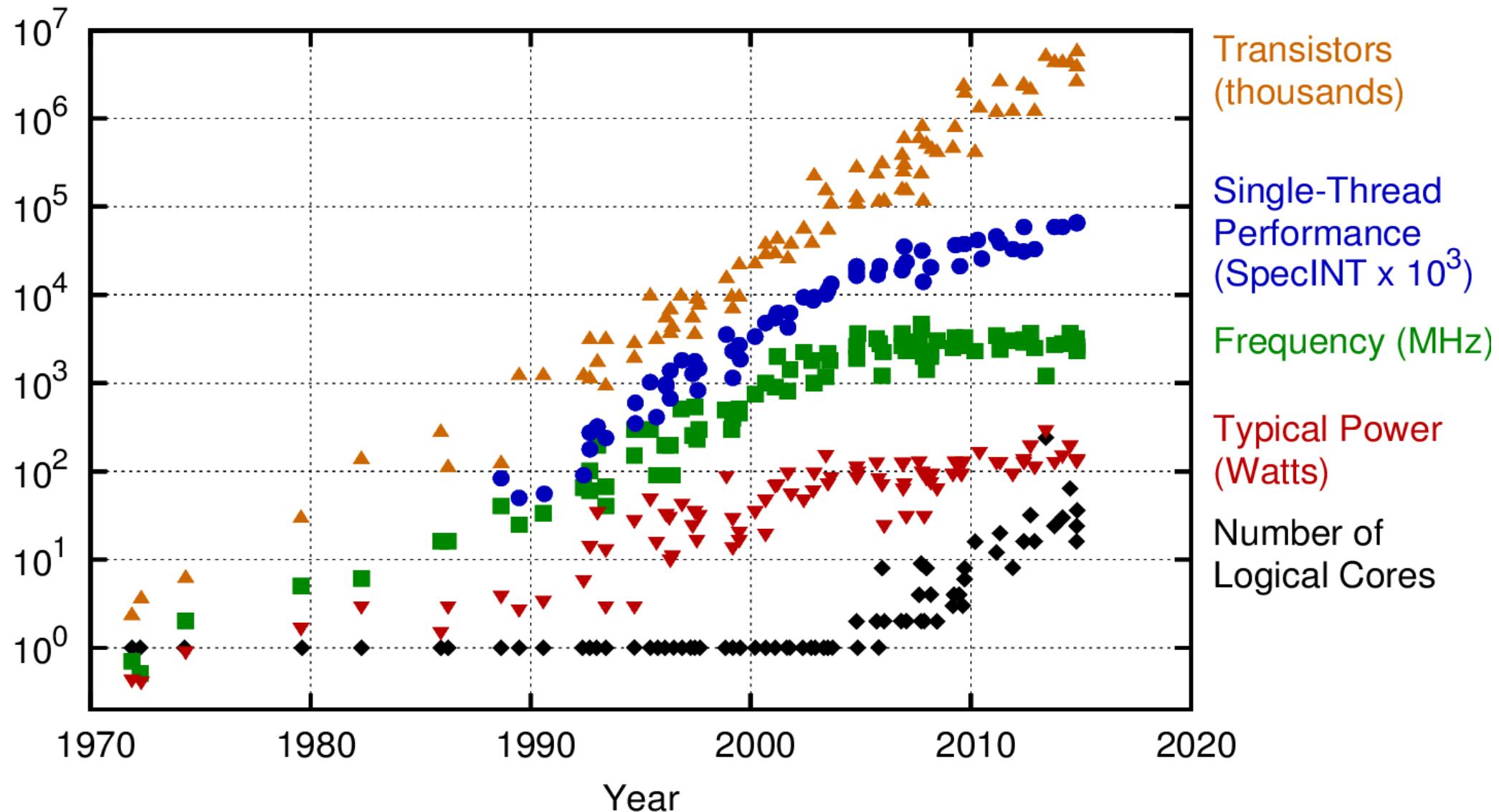
## Loi de Moore (fin des années 60)

Deux fois plus de transistors dans un circuit de même taille tous les 18 mois

## Déclaration de Gordon Moore en 1997

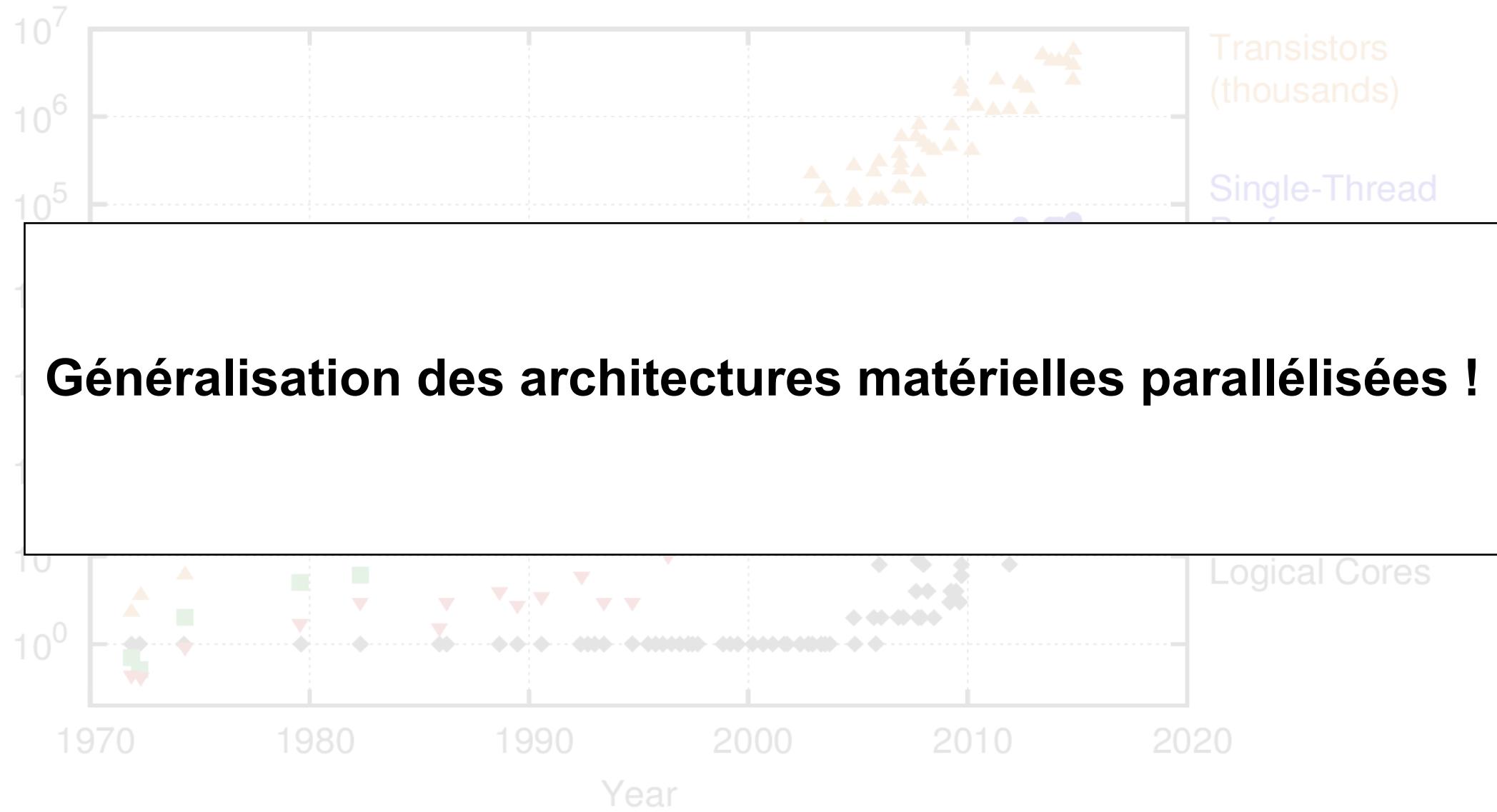
Fin de la loi en 2017 dû à une limite physique.

## 40 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
 New plot and data collected for 2010-2015 by K. Rupp

### 40 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2015 by K. Rupp

# Préambule - Emergence des GPU

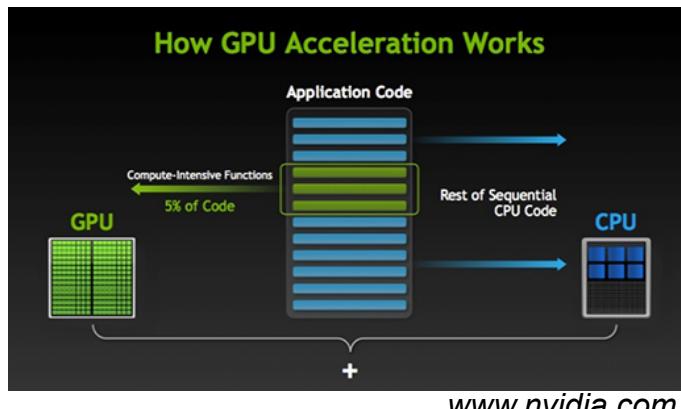
## Années 1990 :



Need for speed 2

- Fort développement des jeux vidéos 3D
- Émergence des cartes graphiques (GPU) dédiées à la 3D
- Cartes spécialisées dans la parallélisation de fonction d'algèbre linéaire (multiplications matrices/vecteurs).

## Années 2000 :



- 2001 : Ouverture aux programmateurs du pipeline graphique des cartes Nvidia.
  - Développement du calcul GPGPU (General Purpose GPU).
- 2007 : Lancement du langage CUDA chez Nvidia
  - Ouvre clairement la possibilité de largement paralléliser des calculs sur GPU.

2009 : Lancement d'OpenCL

- concurrent *libre* de CUDA qui fonctionne sur multiples plateformes (GPU Nvidia, ATI, AMD; CPU INTEL)

2014 - ... : Roc-M (AMD), Metal (Apple), ...

## Ces dernières années :

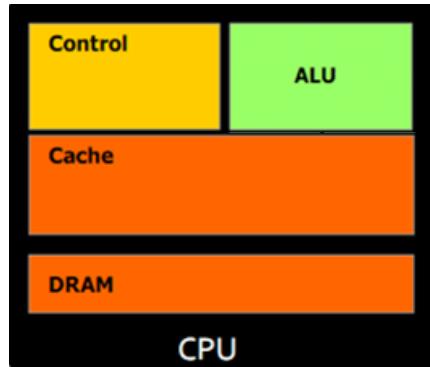
- Utilisation courante du calcul GPU en calcul intensif (HPC)
- Succès du Deep Learning et de XGBoost fortement lié au calcul GPGPU

## **PREAMBULE**

**PARTIE 1 : Principes du calcul GPGPU**

**PARTIE 2 : En pratique**

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

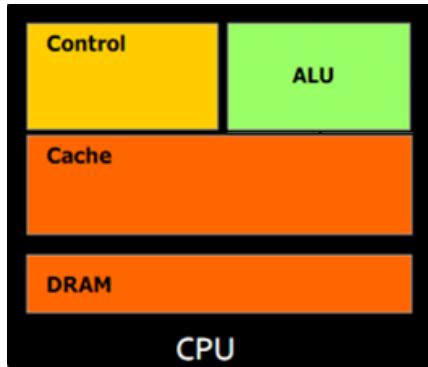
**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

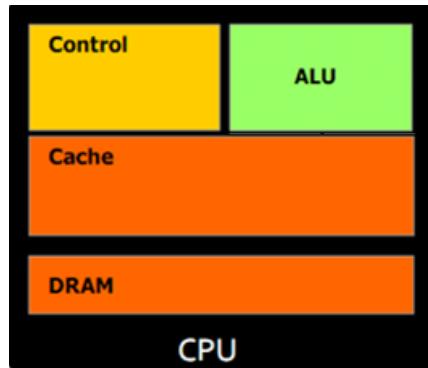
**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{pmatrix}$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mly.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

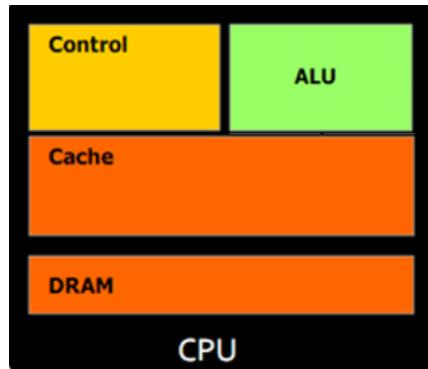
## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{pmatrix}$$

Structuration possible dans la DRAM ou le Cache

$$1 \ 3 \ \dots \ -2 \quad 2 \ 1 \ \dots \ 0 \quad \dots \quad 0 \ 1 \ \dots \ 3 \quad 1 \ -1 \ 0 \ \dots \ 1 \quad ? \ ? \ ? \ \dots \ ?$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

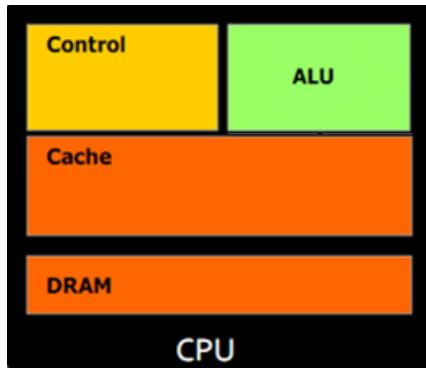
## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{pmatrix}$$

Opérations + et \* sur différentes zones mémoires par le CPU

$$\boxed{1} \ 3 \ \dots \ -2 \ 2 \ 1 \ \dots \ 0 \ \dots \ 0 \ 1 \ \dots \ 3 \ \boxed{1} \ -1 \ 0 \ \dots \ 1 \ \boxed{?} \ ? \ ? \ \dots \ ?$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

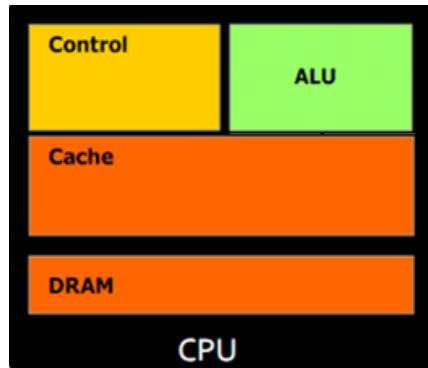
## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & \boxed{3} & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{pmatrix}$$

Opérations + et \* sur différentes zones mémoires par le CPU

$$1 \boxed{3} \dots -2 2 1 \dots 0 \dots 0 1 \dots 3 1 \boxed{-1} 0 \dots 1 \quad \boxed{?} \quad ? \quad ? \dots ?$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

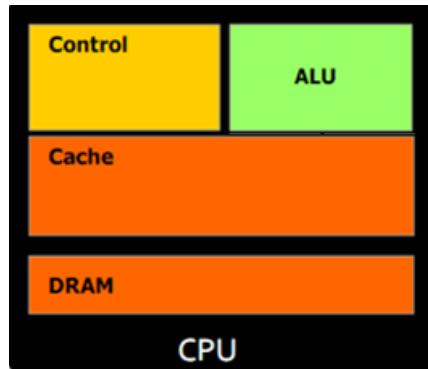
## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & \dots & 2 \\ 2 & 1 & \dots & \dots & 0 \\ -1 & -2 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 1 & \dots & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \\ ? \end{pmatrix}$$

Opérations + et \* sur différentes zones mémoires par le CPU

$$1 \ 3 \ \dots \boxed{-2} \ 2 \ 1 \ \dots \ 0 \ \dots \ 0 \ 1 \ \dots \ 3 \ 1 \ -1 \ 0 \ \dots \boxed{1} \ \boxed{?} \ ? \ ? \ ? \ \dots \ ?$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

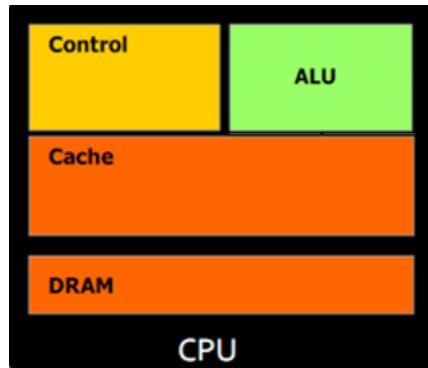
## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ 1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{pmatrix}$$

Opérations + et \* sur différentes zones mémoires par le CPU

$$1 \ 3 \ \dots \ -2 \boxed{2} \ 1 \ \dots \ 0 \ \dots \ 0 \ 1 \ \dots \ 3 \boxed{1} \ -1 \ 0 \ \dots \ 1 \ \boxed{?} \ \boxed{?} \ \dots \ \boxed{?}$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

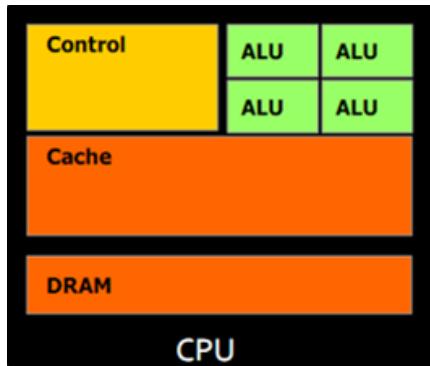
## Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \\ ? \end{pmatrix}$$

Opérations + et \* sur différentes zones mémoires par le CPU

$$1 \ 3 \ \dots \ -2 \ 2 \ 1 \ \dots \ 0 \ \dots \ 0 \ 1 \ \dots \ 3 \ 1 \ -1 \ 0 \ \dots \ 1 \ ? \ ? \ ? \ \dots \ ?$$

# 1) Calcul GPGPU – CPU



<http://igm.univ-mtl.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Unité de calculs.

**Control** : Coordonne les ALU et la mémoire

Illustration : Multiplication matrice / vecteur

$$\begin{pmatrix} 1 & 3 & \dots & \dots & -2 \\ 2 & 1 & \dots & \dots & 0 \\ -1 & -2 & \dots & \dots & 0 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ 0 & 1 & \dots & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{pmatrix}$$

1er pas vers la parallélisation :  
Paradigme *Single Instruction, Multiple Data* (SIMD)

→ L'unité de contrôle demande à plusieurs ALU de faire simultanément le même calcul dans des cases mémoire différentes (notion de WARP)

$$1 \boxed{3 \dots -2} 2 1 \dots 0 \dots 0 1 \dots 3 \boxed{1-1 0 \dots 1} \quad \boxed{\begin{matrix} ? \\ ? \\ ? \\ \vdots \\ ? \end{matrix}}$$

# 1) Calcul GPGPU – GPU

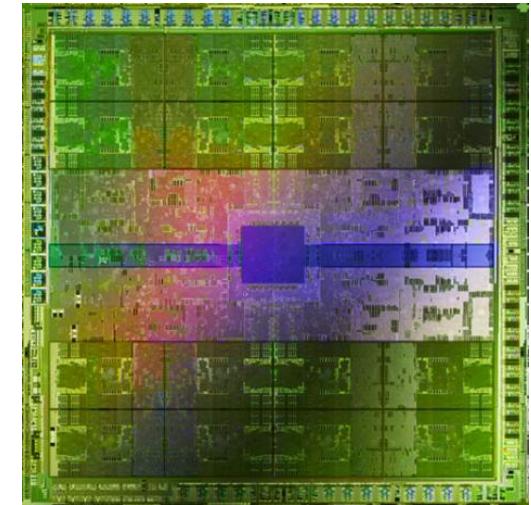
## Intuition naïve (et partiellement fausse)

Un GPU donne accès à des centaines de processeurs qui vont traiter de manière parallèle mes données en mémoire, d'où un gain évident en temps de calculs.

## Réalité matérielle

Modèle de mémoire complexe dû à :

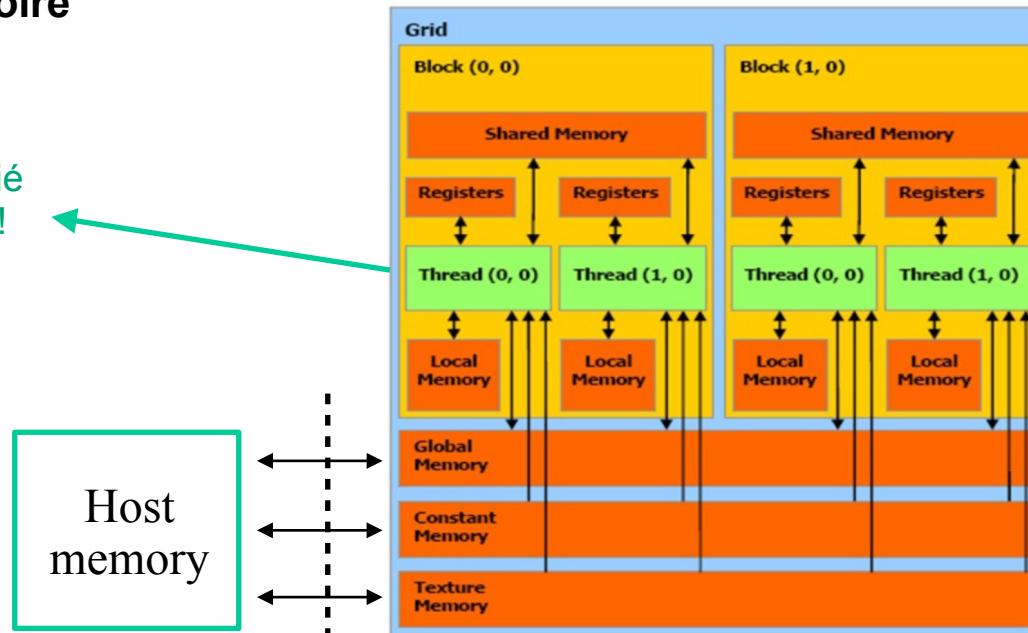
- Contraintes physiques
- Coût des différents niveaux de mémoires (caches L1, L2 et L3)



NVIDIA® Tesla® C2090

## Modèle de mémoire

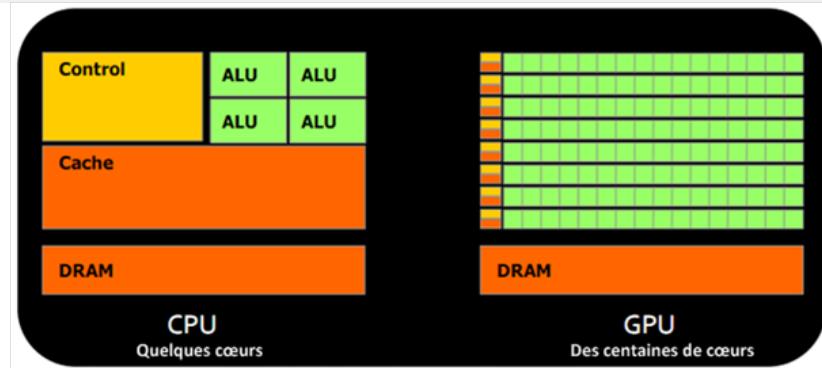
Chaque thread est lié  
à une ALU du GPU !



Mémoire  
classique  
(RAM)

Host  
memory

# 1) Calcul GPGPU – GPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

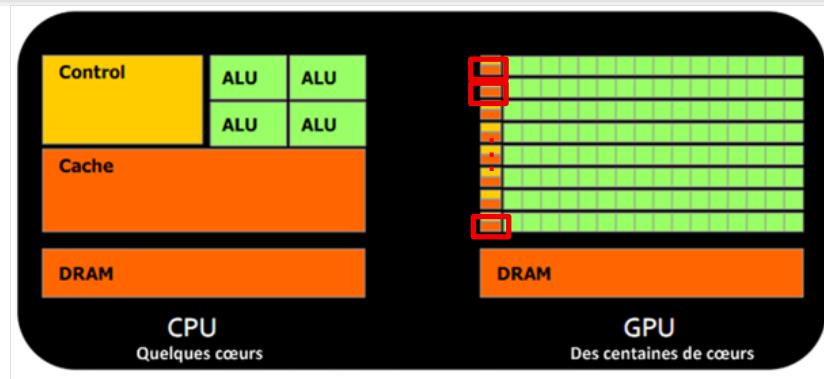
**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Effectue les calculs.

**Control** : Coordonne les ALU et la mémoire

$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \\ ? \end{pmatrix} \xrightarrow{\text{DRAM}} 1\ 3\dots-2\ 2\ 1\dots0\dots0\ 1\dots3\ 1-1\ 0\dots1\ 1\dots1$$

## 1) Calcul GPGPU – GPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

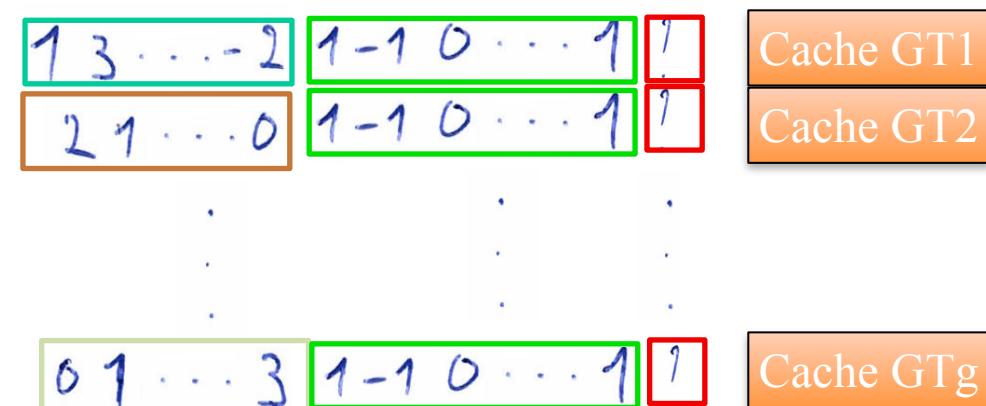
**ALU** : Arithmetic-Logic Unit. Effectue les calculs.

**Control** : Coordonne les ALU et la mémoire

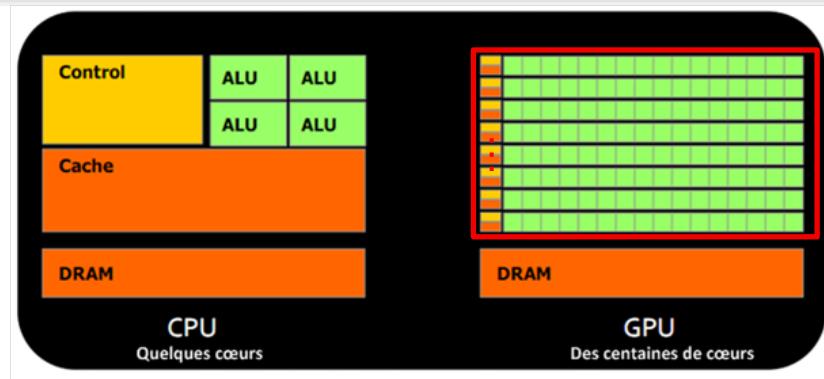
$$\begin{pmatrix} 1 & 3 & \dots & -2 \\ 2 & 1 & \dots & 0 \\ -1 & -2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \\ ? \end{pmatrix}$$

DRAM

**Etape 1** : copies des données dans les différents groupes de travail et allocation mémoire (plus gestion de flux)



## 1) Calcul GPGPU – GPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

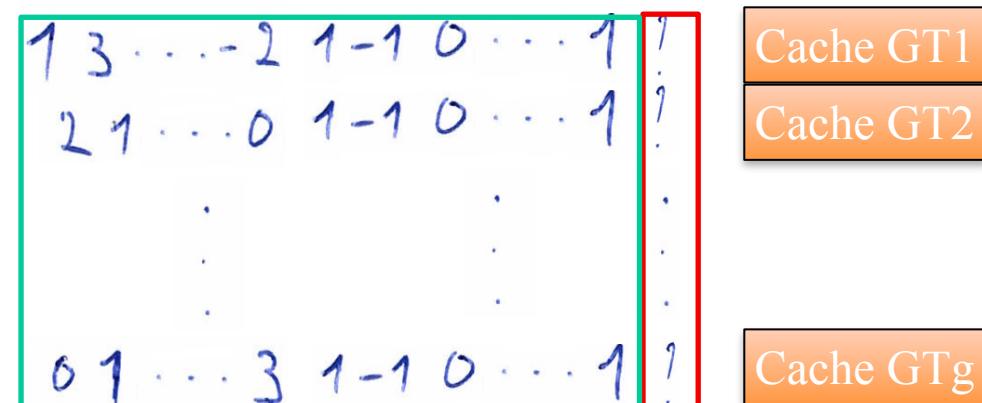
**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Effectue les calculs.

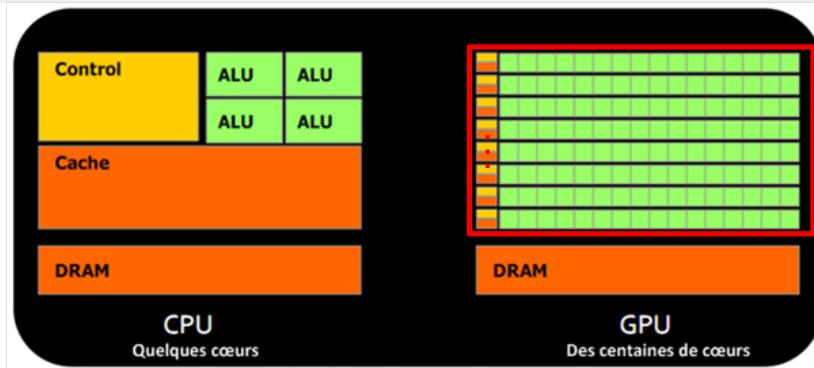
**Control** : Coordonne les ALU et la mémoire

$$\left( \begin{array}{cccccc} 1 & 3 & \dots & \dots & -2 \\ 2 & 1 & \dots & \dots & 0 \\ -1 & -2 & \dots & \dots & 0 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ 0 & 1 & \dots & \dots & 3 \end{array} \right) \left( \begin{array}{c} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{array} \right) = \left( \begin{array}{c} ? \\ ? \\ ? \\ \vdots \\ ? \\ ? \end{array} \right) \xrightarrow{\text{DRAM}} 13\dots-221\dots0\dots01\dots31-10\dots1\ 111\dots1$$

**Etape 2 :** Toutes les multiplications et additions sont faites en parallèle



# 1) Calcul GPGPU – GPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

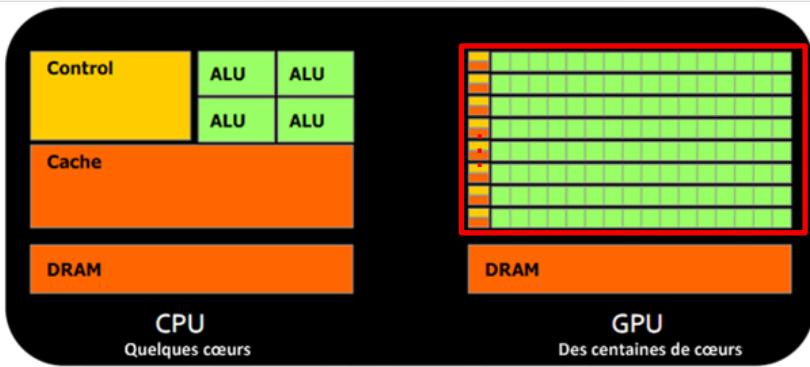
**ALU** : Arithmetic-Logic Unit. Effectue les calculs.

**Control** : Coordonne les ALU et la mémoire

$$\begin{pmatrix} 1 & 3 & \dots & \dots & -2 \\ 2 & 1 & \dots & \dots & 0 \\ -1 & -2 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 1 & \dots & \dots & 3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \\ \vdots \\ ? \\ ? \end{pmatrix} \xrightarrow{\text{DRAM}} 13\dots-221\dots0\dots01\dots31-10\dots1 \quad \boxed{111\dots1}$$

### **Etape 3 : Résultat copié dans la DRAM**

# 1) Calcul GPGPU – GPU



<http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU>

**DRAM** : Dynamic Random Access Memory (mémoire classique)

**Cache** : Mémoire (ici) interne au processeur. Rapide mais de taille limité

**ALU** : Arithmetic-Logic Unit. Effectue les calculs.

**Control** : Coordonne les ALU et la mémoire

## Réflexions pour la programmation haut-niveau :

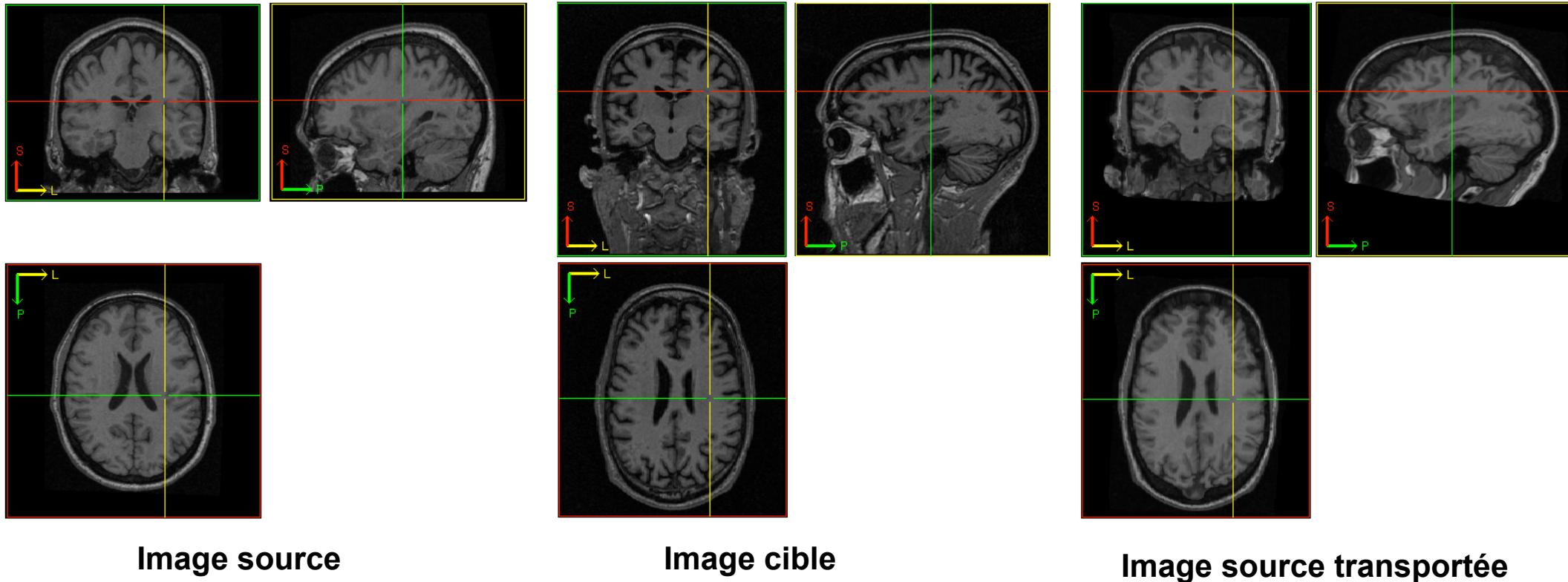
- Est-ce que les mêmes instructions (additions, multiplications, ...) vont s'effectuer sur de gros blocs de données ? ( → notion de divergence )
- Est-ce que ces blocs de données sont continus en mémoire ? ( → notion de coalescence )

## Réflexions pour la programmation bas-niveau :

- [Gain de temps en parallélisant les calculs] - [perte de temps dû aux transferts de données]  $> 0$  ?
- Peut-on rendre négligeable les temps de transferts ? ( → notion de latence )

# 1) Calcul GPGPU – CPU vs GPU

Recalage d'images médicales 3D avec [Vialard et al., IJCV 2012] :



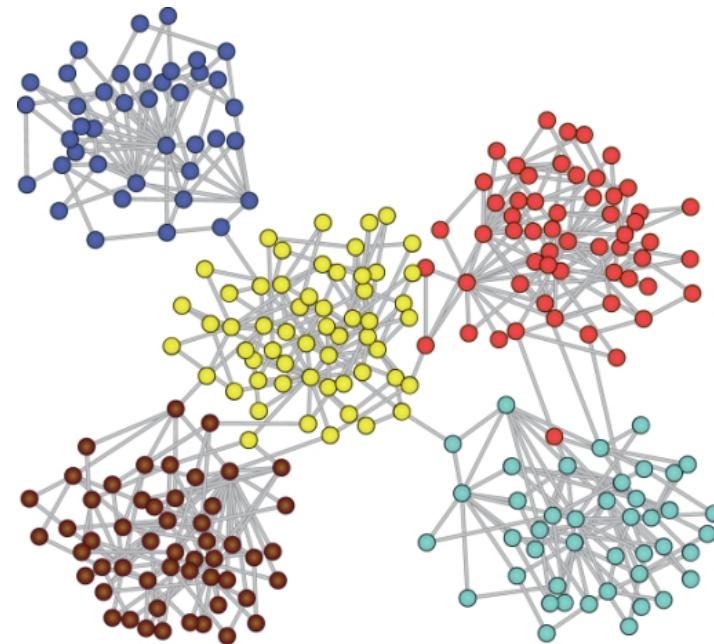
Taille des images	Programme d'origine	Nvidia GTX 780	Intel Iris Graphics 6100
$100 \times 100$	213s	27s	50s
$48 \times 48 \times 48$	58s	1.3s	4.1s
$192 \times 192 \times 192$	1h 51min	2min 8s	17min 38s

(Tests effectués avec OpenCL en 2019)

# 1) Calcul GPGPU – CPU vs GPU

Partitionnement de graphes principalement issus du 9th DIMACS Challenge ([www.diag.uniroma1.it/challenge9/](http://www.diag.uniroma1.it/challenge9/))

- 1) Graphes sociaux et routiers de grande taille
- 2) Algorithme agglomératif basé sur la modularité [Newman, PNAS 2006]



Exemple de partitionnement de graphe (<https://openi.nlm.nih.gov>)

Computation time (s)	#of clusters	CPU	GPU1	GPU2	GPU3
FACEBOOK - $4 \times 10^3$ nodes	5	0.58	0.081	0.09	0.05
	10	0.12	0.080	0.08	0.06
ZBMATHS - $9 \times 10^4$ nodes	10	0.34	0.27	0.32	0.20
	100	1.50	1.26	1.17	0.88
USA_NY - $3 \times 10^5$ nodes	10	1.71	2.39	1.20	1.55
	100	7.33	8.84	4.43	5.12
	1000	X	X	30.12	32.26
USA_FLA - $10^6$ nodes	10	12.43	10.13	3.73	4.47
	100	70.85	51.67	17.01	23.05
USA_LKS - $3 \times 10^6$ nodes	10	55.84	46.77	10.97	13.53
	50	387.93	175.20	28.38	43.77
	100	X	X	48.24	66.69
USA_USA - $2 \times 10^7$ nodes	2	X	X	185.28	238.42
	10	X	X	213.66	X

Architectures testées :

- 1) CPU : Intel Core i5-4200h 2.80 GHz
- 2) GPU1 : NVIDIA GeForce 840M
- 3) GPU2 : NVIDIA GTX 780
- 4) GPU3 : NVIDIA Quadro 5000

(Tests effectués avec OpenCL en 2017)

# 1) Calcul GPGPU – CPU vs GPU

Pour les réseaux de neurones :

2 x 36 threads  
2 x 2200 euros  
en 01/2023

320 tensor cores  
7200 euros en  
01/2023

**Table 2**

**BERT Inference Performance Comparison between Intel CPU and NVIDIA GPU**

	DUAL INTEL XEON GOLD 6240	NVIDIA T4 (TURING)
<b>BERT Inference* Question-Answering (sentences/sec)</b>	2	118
<b>Total Processor TDP</b>	300 W[150x2]	70 W
<b>Energy Efficiency (using TDP)</b>	.007 sentences/sec/W	1.7 sentences/sec/W
<b>GPU Performance Advantage</b>	1.0 (baseline)	59x
<b>GPU Energy-Efficiency Advantage</b>	1.0 (baseline)	240x

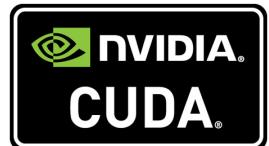
(Tests effectués en 2021 —  
source : [www.curtisswrightds.com](http://www.curtisswrightds.com))

Apprentissage de réseaux de neurones  
RoBERTa sur 400 000 biographies linkedin  
pour la prédiction de 28 métiers avec PyTorch.

Serveur CPU IMT : Intel® Xeon® CPU E7-8890 v4 @ 2.20GHz —192 cœurs , 512 Go RAM → **6 heures**

Serveur GPU ANITI : Nvidia Quadro RTX 8000 — 4608 coeurs, 48Go RAM → **1 heure**

## 2) En pratique – Installation



### Installation de drivers CUDA :

→ <https://developer.nvidia.com/cuda-downloads>

### Compilateurs possibles :

→ gcc/g++, Clang/xCode, Visual Studio, ...

### Librairies d'interfaces PyCuda pour la programmation en Python :

→ <https://mathematician.de/software/pycuda/>

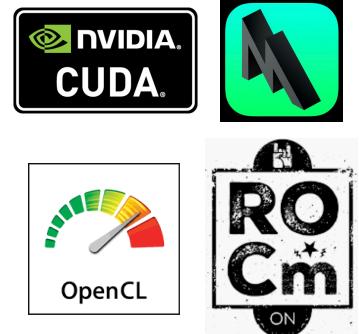
→ <https://anaconda.org/lukefister/pycuda>

→ Installation possible dans des notebooks de Google Colab !

## 2) En pratique – Installation

### Remarques :

- Apple a délaissé *nvidia/cuda* et offre un support limité à *OpenCL*. La solution maison *Metal* est poussée.
- *AMD* pousse aussi sa solution maison *ROCM*.



### En vue de faire des TP's :

- Si le PC a une carte GPU Nvidia, installer en local le nécessaire.
- Sinon, tout marche très bien sur google colab ou kaggle !

A screenshot of a Google Colab notebook titled "TP\_ISAE\_matmul\_cuda.ipynb". The notebook interface includes a toolbar with file, edit, view, insert, run, tools, help, and a "Toutes les..." dropdown. On the left is a sidebar with code and text tabs, search, and file/folder icons. The main area shows a code cell with the following content:

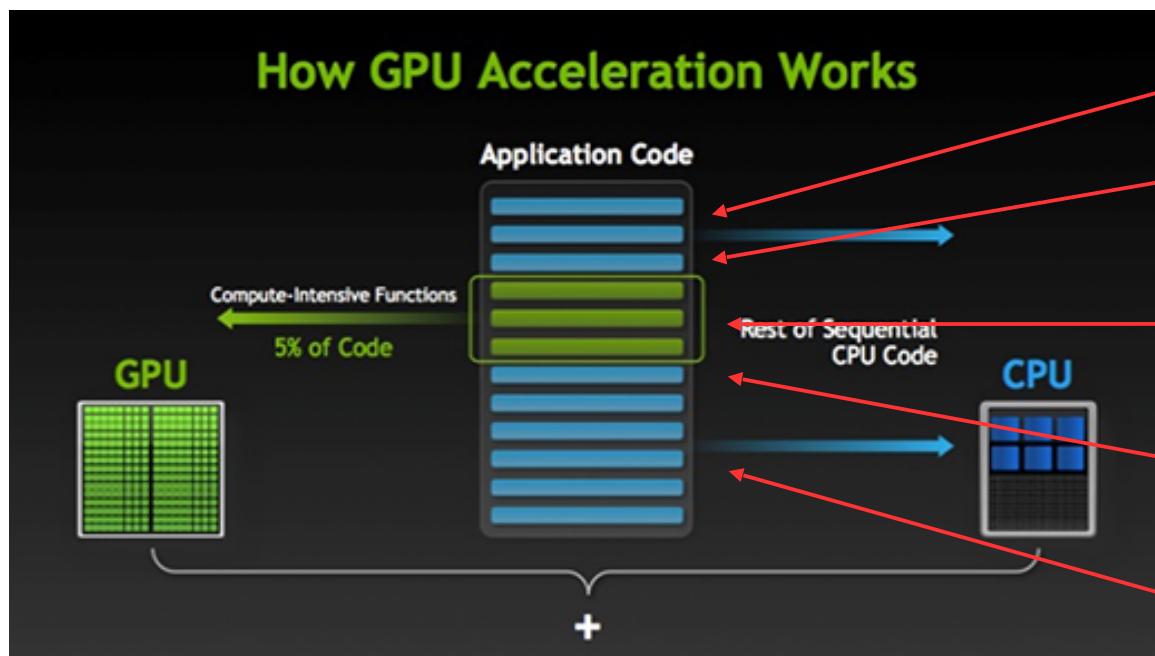
```
#get device information
MyDevice=pycuda.driver.Device(0)
MyDevice.name()

'Tesla T4'

[ ]
#define the kernel
kernel_code_template.....|||||
```

The cell has a play button icon and a timestamp of "0 s". To the right of the code are execution status indicators (green checkmark for RAM, grey for Disque), modification controls, and a toolbar with up/down arrows, copy/paste, and other options.

## 2) En pratique – Exemple de code CUDA appelé par Python



### Dans notre exemple

Code de calcul Python exécuté ;  
Python copie des données dans la mémoire *device* (GPU) ;  
Python appelle une routine CUDA pour un calcul parallélisé sur les données en mémoire *device* ;  
Python copie les résultats dans la mémoire *host* ;  
Suite de l'exécution du code Python.

## 2) En pratique – Exemple de code CUDA appelé par Python

1 : Appel et initialisation de la librairie (API) PyCUDA en début de fichier :

```
import numpy as np
import pycuda.autoinit
from pycuda import driver , gpuarray

from pycuda.compiler import SourceModule
```

2 : Définition et compilation d'une fonction (kernel) qui s'applique en **un point** d'un vecteur :

```
my_kernel="""
__global__ void addition(float *a, float *b, float *result)
{
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    result[index] = a[index] + b[index];
}
"""

mod = SourceModule(my_kernel)

addition = mod.get_function("addition")
```

## 2) En pratique – Exemple de code CUDA appelé par Python

### 3 : Transferts DRAM / mémoire GPU et appel de la fonction

```
a_cpu = np.random.randn(1024).astype(np.float32)
b_cpu = np.random.randn(1024).astype(np.float32)
```

Définition des arrays dans la mémoire classique (host)

```
# transfer host (CPU) memory to device (GPU) memory
a_gpu = gpuarray.to_gpu(a_cpu)
b_gpu = gpuarray.to_gpu(b_cpu)

# create empty gpu array for the result (C = A + B)
c_gpu = gpuarray.empty((1024), np.float32)
```

Copie des arrays dans la mémoire GPU (device) et allocation d'une zone mémoire supplémentaire pour le résultat

```
addition(a_gpu , b_gpu , c_gpu , block = (1024, 1, 1))
```

Appel de la fonction CUDA

```
c_cpu = gpuarray.GPUArray.get(c_gpu)
```

Copie du résultat dans la mémoire classique

```
print('a = ',a_cpu)
print('b = ',b_cpu)
print('c = ',c_cpu)
```

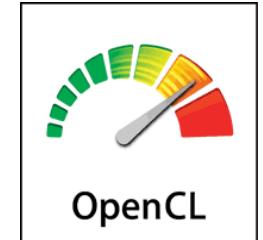
Vérification du résultat

```
a = [ 1.0868498 -1.161501 -0.20488106 ... -0.65797645 -0.91217256
      1.4827517 ]
b = [ 0.14378278  1.8578753 -0.68487275 ...  2.043415     0.05926542
      -0.82302237]
c = [ 1.2306325   0.6963743 -0.8897538 ...  1.3854387 -0.8529071
      0.65972936]
```

## 2) En pratique – Librairies généralistes utilisant du calcul GPGPU

### Algèbre linéaire :

- 1) clBLAS (BLAS niveau 1, 2, 3 – openCL)
- 2) cuBLAS (BLAS niveau 1, 2, 3 – CUDA)
- 3) ViennaCL (openCL)
- 4) VexCL (openCL/CUDA)
- 5) CUSP (CUDA)
- 6) CULA dense et sparse (Lapack et BLAS – CUDA)
- 7) ...



### FFT :

- 1) CUFFT (CUDA)
- 2) clFFT (openCL)
- 3) ...



VexCL

$\exists$ iennaCL

### Génération de nombres aléatoires :

- 1) cuRAND (CUDA)
- 2) clRNG (openCL)

C U S P

**c**ULA|tools

### Numérique généraliste :

- 1) ArrayFire (CUDA et OpenCL)
- 2) ...

cuFFT

clFFT  
clRNG



{ } ARRAYFIRE

# NVIDIA CUDA-X

## GPU-Accelerated Libraries

NVIDIA® CUDA-X, built on top of NVIDIA CUDA®, is a collection of libraries, tools, and technologies that deliver dramatically higher performance—compared to CPU-only alternatives—across multiple application domains, from artificial intelligence (AI) to high performance computing (HPC).

NVIDIA libraries run everywhere from resource-constrained IoT devices, to self-driving cars, to the largest supercomputers on the planet. As a result, you get highly-optimized implementations of an ever-expanding set of algorithms. Whether you're building a new application or accelerating an existing application, NVIDIA libraries provide the easiest way to get started with GPU acceleration.

## Components



Math Libraries



Parallel Algorithms



Image and Video Libraries



Communication Libraries



Deep Learning



Partner Libraries

## 2) En pratique – Librairies généralistes de Nvidia

### Math Libraries

GPU-accelerated math libraries lay the foundation for compute-intensive applications in areas such as molecular dynamics, computational fluid dynamics, computational chemistry, medical imaging, and seismic exploration.



#### cuBLAS

GPU-accelerated basic linear algebra (BLAS) library

[Learn More](#)



#### cuFFT

GPU-accelerated library for Fast Fourier Transforms

[Learn More](#)



#### CUDA Math Library

GPU-accelerated standard mathematical function library

[Learn More](#)



#### cuRAND

GPU-accelerated random number generation (RNG)

[Learn More](#)



#### cuSOLVER

GPU-accelerated dense and sparse direct solvers

[Learn More](#)



#### cuSPARSE

GPU-accelerated BLAS for sparse matrices

[Learn More](#)



#### cuTENSOR

GPU-accelerated tensor linear algebra library

[Learn More](#)



#### AmgX

GPU-accelerated linear solvers for simulations and implicit unstructured methods

[Learn More](#)

# Deep Learning Libraries

GPU-accelerated libraries for Deep Learning applications that leverage CUDA and specialized hardware components of GPUs.

## NVIDIA cuDNN

GPU-accelerated library of primitives for deep neural networks

[Learn More](#)

## NVIDIA TensorRT™

High-performance deep learning inference optimizer and runtime for production deployment

[Learn More](#)

## NVIDIA Riva

Platform for developing engaging and contextual AI-powered conversation apps

[Learn More](#)

## NVIDIA DeepStream SDK

Real-time streaming analytics toolkit for AI-based video understanding and multi-sensor processing

[Learn More](#)

## NVIDIA DALI

Portable, open-source library for decoding and augmenting images and videos to accelerate deep learning applications

[Learn More](#)

## 2) En pratique – Librairies de machine learning utilisant du calcul GPGPU

Deep learning

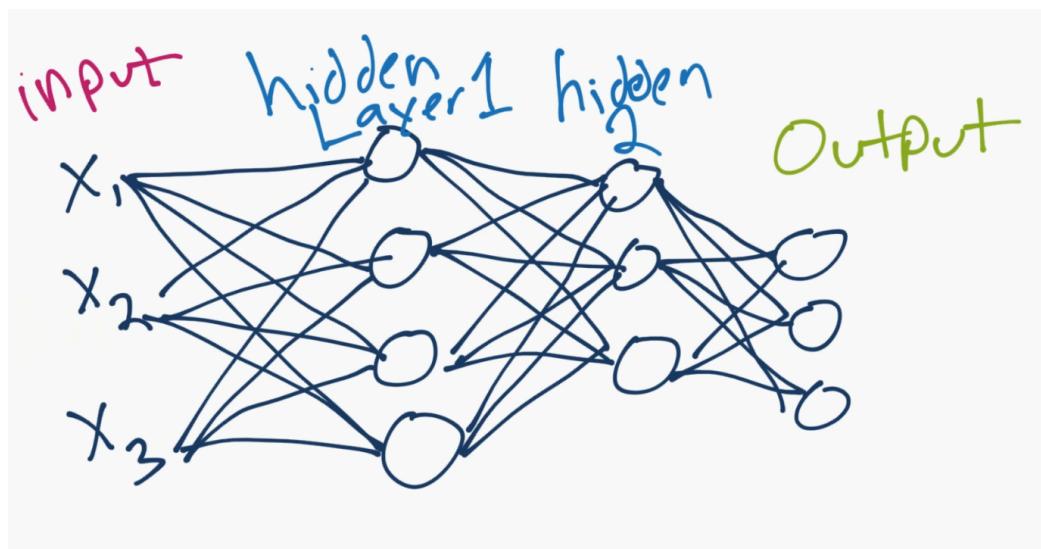
Caffe

Keras

theano

PYTORCH

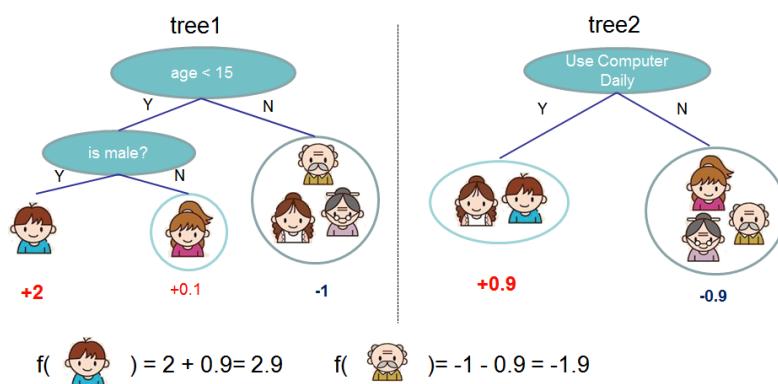
...



<https://pythonprogramming.net/neural-networks-machine-learning-tutorial/>

XGBoost

**XGBoost**



- 1) Très efficace sur certains problèmes (signaux, images).
- 2) Apprentissage nécessitant du calcul intensif mais prédiction rapide.
- 3) Dimension du problème d'apprentissage très élevée.
- 4) Nécessité d'avoir énormément de données annotées ou une structure de graphe adaptée aux données.

- 1) Méthode basée sur des arbres de décision.
- 2) Très efficace dans le cas général.
- 3) Paramètre à gérer (contrairement à Random Forest) et gros coût calculatoire.

- 1) Extrêmement efficace au moins pour l'imagerie, les réseaux de neurones et le calcul matriciel.
- 2) Nécessite un investissement de temps non-négligeable pour s'y mettre.
- 3) Efficacité dépendant de la structure des données en mémoire mais aussi de l'architecture matérielle.
- 4) Il existe une multitude de projets liés au calcul GPU.