# Calculs parallélisés sur GPU avec Python

## Les librairies Pytorch et Numba

Olivier Goudet

SFR MathStic

21 octobre 2022

# Plan

1. Processeur graphique (GPU)
2. Calculs tensoriels avec la librairie Pytorch
3. Implémentation de *kernels* avec la librairie Numba
4. Présentation de ressources régionales et nationales pour le calcul GPU

.

Section 1

# Processeur GPU

# Processeur graphique (GPU)

- GPU : *Graphics Processing Unit*
- Structure hautement parallèle qui le rend efficace pour une large palette de tâches graphiques comme le rendu 3D, le traitement du signal vidéo, etc...
- Intéressant pour le calcul matriciel, le deep learning, etc.
- Marques de GPU : NVIDIA, AMD et Intel.

# Carte Nvidia V100

NVIDIA Tesla V100 32GB Graphic Card, CUDA Cores 5120

NVIDIA · Tesla · Cartes graphiques · 32 go de mémoire vidéo

NVIDIA Tesla V100 900-2G500-0010-000 Tesla V100 32GB Graphic Card TV100GC

Plus de détails sur le site de B&H Photo-Video-Audio »

**8 954,82 € + taxes**
9 900,30 $US + taxes
+ 68,75 € de frais de port
B&H Photo-Video-Audio

Visiter le site

# Comparaison CPU vs GPU

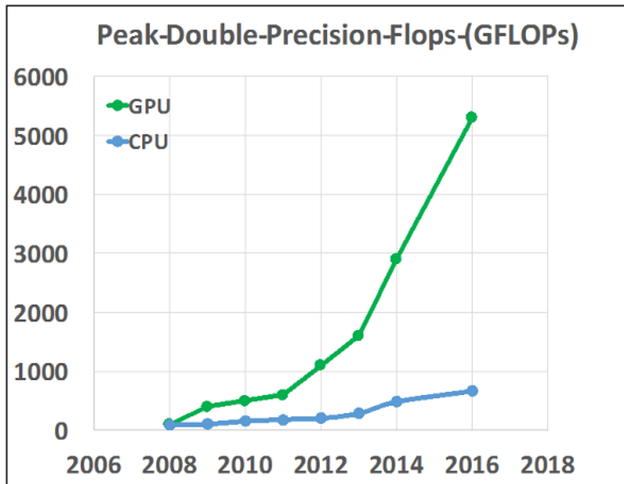
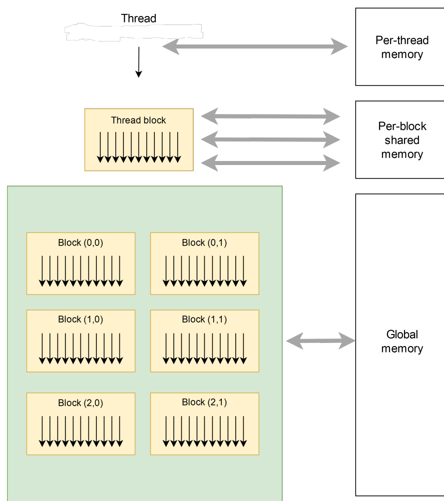
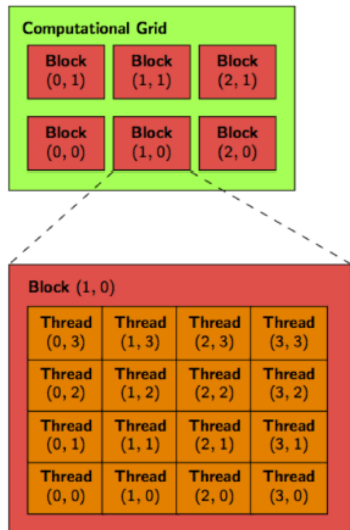



Figure 1: source : https://www.hpcwire.com/2016/08/23/2016-important-year-hpc-two-decades/

# Architecture GPU

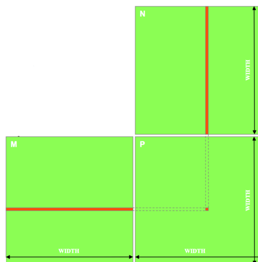


# Grille de calcul sur un GPU



# Exemple de calcul matriciel sur un GPU

- Produit matricielle P de deux matrices carrés M et N, de taille $(n, n)$.
- Le calcul de chacune des valeurs de P est un calcul indépendant.
- Un thread pour le calcul de chaque valeur de P.



# Utilisation d'un GPU couplé à un CPU

- GPU utilisé généralement comme "co-processeur" de calcul pour le CPU de la machine hôte.
- Mémoire du GPU distincte de celle du CPU.
- On peut faire des copies de l'une à l'autre (coûteux).

# Programmation GPU

Plusieurs possibilités :

- **Programmation "haut niveau"** : calcul matriciel ou tensoriel.
  $\rightarrow$ e.g. librairie Pytorch.
- **Programmation "plus bas niveau"** : implémentions de fonctions noyau (*kernel*) déployées sur la grille de calcul du GPU.
  $\rightarrow$ Cuda C++ ou Numba.

Section 2

# Calcul tensoriel sur GPU avec la librairie Pytorch

# Manipulation de tenseurs avec Pytorch

- Manipulation de tenseurs avec Pytorch similaire à la manipulation de Numpy Arrays.
- Certaines fonctions changent toutefois un peu de syntaxe par rapport à Numpy, mais reste très semblables.
- On peut facilement convertir un Numpy Array en tenseur Torch et réciproquement.
- Documentation de Pytorch : `https://pytorch.org/`.

# Création d'un tenseur Pytorch

- Exemple de création d'un tenseur Torch de taille (5,3) contenant uniquement des zéros :

```
import torch as th
x = th.zeros((5,3))
print(x)
```

Affiche :

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
```

- On voit que c'est très similaire à la commande pour créer un array de taille (5,3) avec numpy : $x = np.zeros((5,3))$.

# Passage d'un tenseur sur la carte graphique

- Exemple de passage sur la carte graphique nommée "cuda:0".

```
x = th.zeros(5,3)
x = x.to("cuda:0")
print(x)
```

Affiche :

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]], device="cuda:0")
```

# Rapatriement d'un tenseur de la carte graphique vers le cpu

- Exemple de rapatriement vers le cpu.

```
x = x.to("cpu")
print(x)
```

Affiche :

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
```

# Opération entre tenseurs de différents environnements

- Attention les tenseurs doivent se situer dans le même espace mémoire pour pouvoir faire des opérations.
- On ne peut pas faire une opération entre un tenseur situé sur le CPU et un tenseur situé sur le GPU (ou bien sur deux GPU différents).

```
x = th.ones(5,3)
x = x.to("cuda:0")
print(x)
y = th.ones(3,10)
z = x @ y # produit matriciel entre x et y
print(z)
```

Affiche :

RuntimeError: Expected object of backend CUDA but got backend CPU for argument #2 "mat2"

# Implémentation d'un réseau de neurones en Pytorch

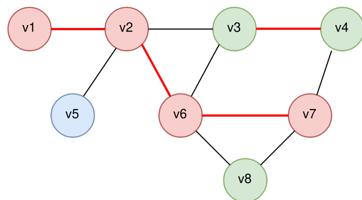
```
class nn_mnist(th.nn.Module):

    def __init__(self,d,k,h1,h2):
        super(nn_mnist, self).__init__()

        self.layer1 = th.nn.Linear(d, h1) # Applique une transformation y = W @ x + b
        self.layer2 = th.nn.Linear(h1, h2)
        self.layer3 = th.nn.Linear(h2, k)

    def forward(self, x):

        phi1 = th.sigmoid(self.layer1(x))
        phi2 = th.sigmoid(self.layer2(phi1))

        return th.softmax(self.layer3(phi2),1)
```

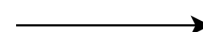
# Entraînement du réseau de neurones sur le GPU

```
device = "cuda:0"
nnet = nn_mnist(28*28,10,200,100).to(device)

optimizer = optim.SGD(nnet.parameters())

for epoch in range(10000):

    for images, labels in trainloader:

        images = images.to(device);
        labels = labels.to(device)

        optimizer.zero_grad()

        output = nnet(images)

        loss = th.nn.CrossEntropyLoss(output,labels)

        loss.backward()

        optimizer.step()
```

# Combinatorial optimization with Pytorch library

- Candidate solution with $k$ color groups : $S = \{s_{i,j}\}$ in $\{0,1\}^{n \times k}$, where $s_{i,j} = 1$ if the $i$-th vertex $v_i$ belongs to the color group $j$ and $s_{i,j} = 0$ otherwise.



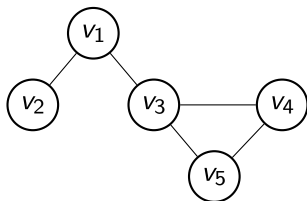



$S$

| | color 0 | color 1 | color 2 |
|---|---|---|---|
| v1 | 1 | 0 | 0 |
| v2 | 1 | 0 | 0 |
| v3 | 0 | 1 | 0 |
| v4 | 0 | 1 | 0 |
| v5 | 0 | 0 | 1 |
| v6 | 1 | 0 | 0 |
| v7 | 1 | 0 | 0 |
| v8 | 0 | 1 | 0 |

# Adjacency matrix of the graph

- For a graph $G = (V, E)$ with $n$ vertices, let $A = \{a_{i,j}\}_{i,j=1...n}$ be its adjacency matrix so that $\{v_i, v_j\} \in E$ if and only if $a_{i,j} = 1$.




$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \quad (1)$$

# Fitness evaluation and gradient

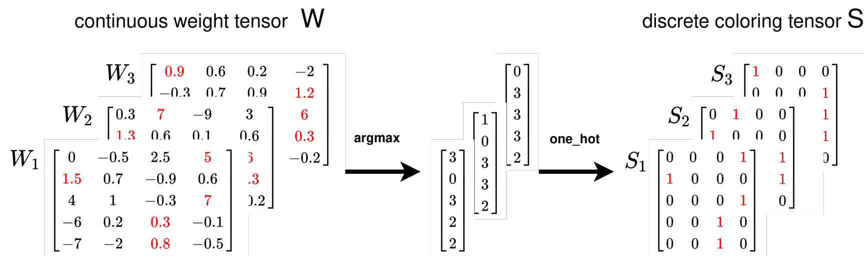
- Using $A$, the fitness of of $S$ can directly be computed as:

$$f(S) = Tr(S' \cdot A \cdot S) \tag{2}$$

- Gradient of f, w.r.t $S$, a matrix of size $n \times k$:

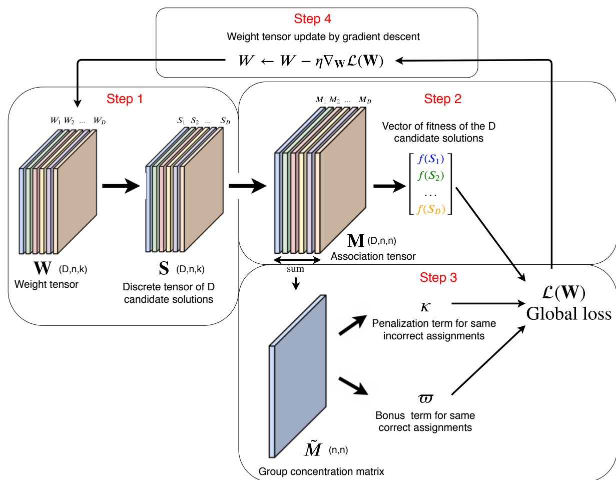
$$\nabla_S f(S) = A \cdot S \tag{3}$$

# Weight formulation of a coloring problem




$$S = g(W) = \text{one_hot}(\text{argmax}(W))$$

# TensCol algorithm : general scheme




Population-based gradient descent weight learning for graph coloring problems. O Goudet, B Duval and Jin-Kao Hao. Knowledge-Based Systems, 212, 106581. 2021.

# Implémentation d'un générateur de colorations de graphes

```
class graphColoringGenerator(th.nn.Module):


    def __init__(self, size, sigma=0.01):
        super(graphColoringGenerator, self).__init__()
        self.weights = th.nn.Parameter(th.FloatTensor(*self.size))
        self.weights.data.normal_(0, sigma)

    def forward(self):

        sample_soft = th.softmax(self.weights, dim=2)

        _, k = sample_soft.data.max(-1)
        logits = self.weights.data
        shape = logits.size()
        sample_hard = logits.data.new(*shape).zero_().scatter_(-1, k.view(shape[0],
             shape[1], 1), 1.0)

        sample = sample_hard.data - sample_soft.data + sample_soft

        return sample
```

# TensCol algorithm in Pytorch

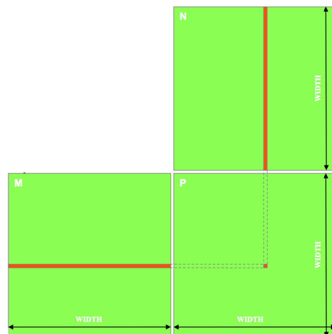
```
device = "cuda:0"
color_sampler = graphColoringGenerator( n, k,sigma_0).to(device)
optimizer = th.optim.Adam(list(color_sampler.parameters()))

for t in range(10000):

    optimizer.zero_grad()
    S = graph_sampler()
    M = S @ S.transpose(1, 2)
    f = th.sum(A * M, dim=[1, 2]) / 2.0
    V_bar = th.sum(V,0)
    sum_fitness = th.sum(f)
    kappa = lambda_*th.sum(A*V_bar**alpha)
    varpi = mu_ * th.sum((II - A) * V_bar)
    sum_l2_regul = l2_regul * th.sum(color_sampler.weights**2)

    Loss = th.sum(f) + kappa - varpi + sum_l2_regul

    Loss.backward()
    optimizer.step()
```

Section 3

# Implémentation de kernels Cuda avec la librairie Numba

# Implémentation naïve d'un kernel pour la multiplication de matrices

```
@cuda.jit
def matmul(A, B, C):

    i, j = cuda.grid(2)

    if i < C.shape[0] and j < C.shape[1]:

        tmp = 0.

        for k in range(A.shape[1]):
            tmp += A[i, k] * B[k, j]

        C[i, j] = tmp
```

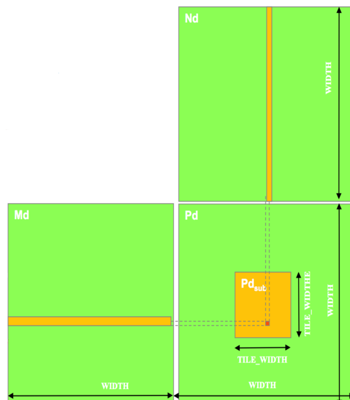


# Utilisation du kernel Numba sur le GPU

```
x_h = np.arange(16384).reshape([128, 128])
y_h = np.ones([128, 128])
z_h = np.zeros([128, 128])

x_d = cuda.to_device(x_h)
y_d = cuda.to_device(y_h)
z_d = cuda.to_device(z_h)

blockspergrid = (1, 1)
threadsperblock = (128, 128)

matmul[blockspergrid, threadsperblock](x_d, y_d, z_d)

z_h = z_d.copy_to_host()
```

# Implémentation plus optimisée (calcul par blocs)

```
@cuda.jit
def fast_matmul(A, B, C):

    sA = cuda.shared.array(shape=(16, 16), dtype=float32)
    sB = cuda.shared.array(shape=(16, 16), dtype=float32)

    x, y = cuda.grid(2)
    tx = cuda.threadIdx.x
    ty = cuda.threadIdx.y
    bpg = cuda.gridDim.x

    tmp = float32(0.)

    for i in range(bpg):

        sA[tx, ty] = 0
        sB[tx, ty] = 0

        if x < A.shape[0] and (ty + i * TPB) < A.shape[1]:
            sA[tx, ty] = A[x, ty + i * TPB]

        if y < B.shape[1] and (tx + i * TPB) < B.shape[0]:
            sB[tx, ty] = B[tx + i * TPB, y]

        for j in range(TPB):
            tmp += sA[tx, j] * sB[j, ty]

    if x < C.shape[0] and y < C.shape[1]:
        C[x, y] = tmp
```

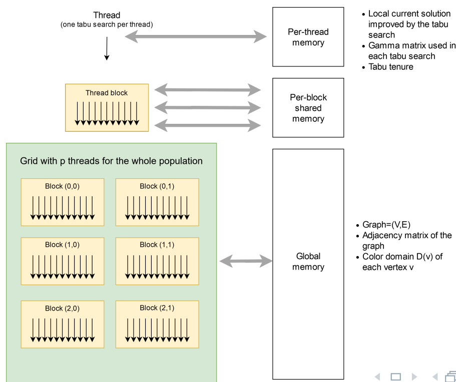


# Utilisation sur le GPU

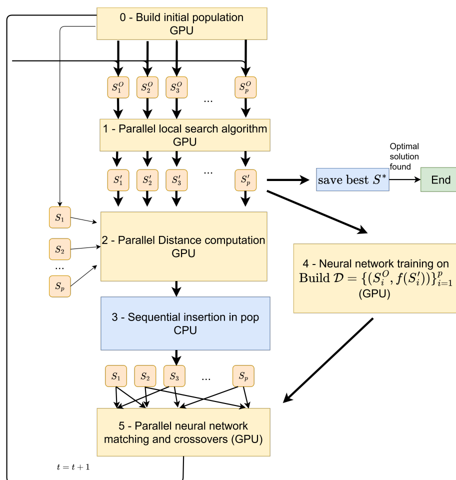
```
x_h = np.arange(16384).reshape([128, 128])
y_h = np.ones([128, 128])
z_h = np.zeros([128, 128])

x_d = cuda.to_device(x_h)
y_d = cuda.to_device(y_h)
z_d = cuda.to_device(z_h)

threadsperblock = (16, 16)
blockspergrid = (8, 8)

matmul[blockspergrid, threadsperblock](x_d, y_d, z_d)

z_h = z_d.copy_to_host()
```

# Mise en oeuvre pour résoudre des problèmes d'optimisation combinatoire

- Algorithme évolutionnaire avec une très grande population (64000 individus).
- Chaque recherche locale est lancée sur un thread du GPU.



# Deep learning guided memetic framework (DLMCOL)




A deep learning guided memetic framework for graph coloring problems. Goudet, O., Grelier, C., & Hao, J. K. (2022). Knowledge-Based Systems, 109986.

Section 4

# Des ressources de calcul GPU disponibles au niveau régional et national

# Centre de Calcul Intensif des Pays de la Loire (CCIPL)

- 22 millions d'heures de calcul par an à la disposition des chercheurs de la région (chiffre de 2017).
- Serveur de calcul CPU et GPU (nouvelles cartes graphiques A100).
- Gestionnaire de batch SLURM.
- Différentes partitions pour des calculs plus ou moins longs.

# Accès au CCIPL

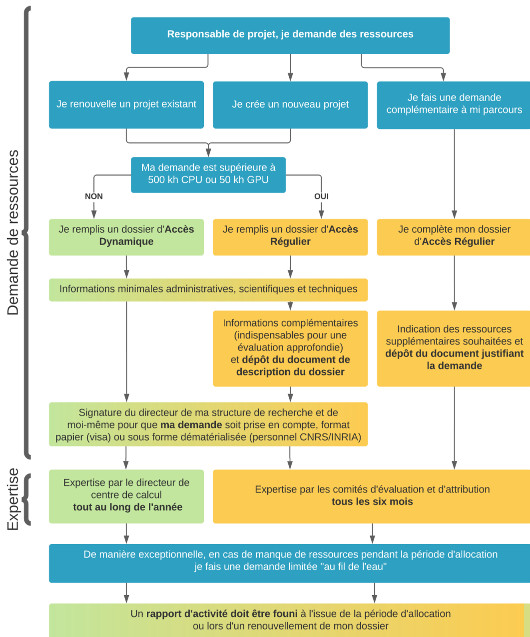
- Formulaire de demandes à rédiger avec présentation du projet : `https://ccipl.univ-nantes.fr/conditions-dacces-et-formulaires-de-demandes`.
- Accès gratuit et permanent.
- Limite d'utilisation simultanée des ressources GPU (e.g. max 8 cartes A100 en même temps).
- Pas de décompte des heures d'utilisation.

# Supercalculateur National Jean Zay

- Société GENCI (Grand équipement national de calcul intensif).
- Installé à l'IDRIS (Orsay), centre national de calcul du CNRS, depuis 2019.
- Permet de lancer des calculs à grande échelle (sur plusieurs centaines de V100 en même temps).

# Accès au serveur Jean Zay

- Formulaires de demandes à rédiger via le portail eDARI : `https://www.edari.fr/`.
- Conditions d'accès plus strictes.
  - Rédiger un projet détaillé.
  - Décompte des heures d'utilisation.
  - Déposer un rapport d'activités à la fin de la période d'attribution des ressources.
- Différents types d'allocations :
  - Accès Dynamique ($<$ 50 kh GPU).
  - Accès Régulier ($>$ 50 kh GPU).


Demande de ressources
Expertise
Responsable de projet, je demande des ressources
Je renouvelle un projet existant
Je crée un nouveau projet
Je fais une demande complémentaire à mi parcours
Ma demande est supérieure à 500 kh CPU ou 50 kh GPU
NON
OUI
Je remplis un dossier d'Accès Dynamique
Je remplis un dossier d'Accès Régulier
Je complète mon dossier d'Accès Régulier
Informations minimales administratives, scientifiques et techniques
Informations complémentaires (indispensables pour une évaluation approfondie) et dépôt du document de description du dossier
Indication des ressources supplémentaires souhaitées et dépôt du document justifiant la demande
Signature du directeur de ma structure de recherche et de moi-même pour que ma demande soit prise en compte, format papier (visa) ou sous forme dématérialisée (personnel CNRS/INRIA)
Expertise par le directeur de centre de calcul tout au long de l'année
Expertise par les comités d'évaluation et d'attribution tous les six mois
De manière exceptionnelle, en cas de manque de ressources pendant la période d'allocation je fais une demande limitée "au fil de l'eau"
Un rapport d'activité doit être founi à l'issue de la période d'allocation ou lors d'un renouvellement de mon dossier