INSTITUT de MATHEMATIQUES de TOULOUSE

cnrs dépasser les frontières
www.cnrs.fr

Retour d'experience sur l'analyse de données avec carte GPU sous OpenCL

Laurent Risser

Institut de mathématiques de Toulouse
lrisser@math.univ-toulouse.fr

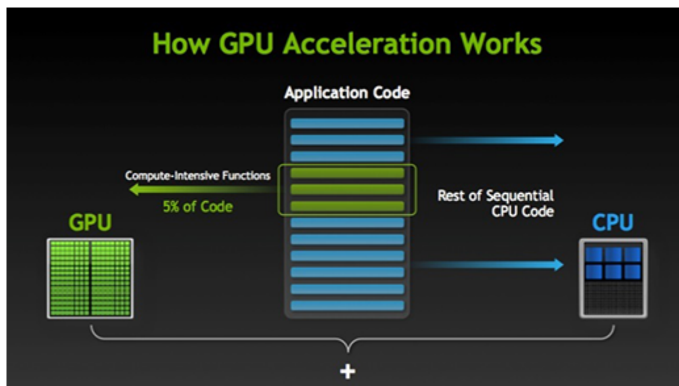
Un peu d’histoire

Années 1990 :

Need for speed 2

- Fort développement des jeux vidéos 3D
- Émergence des cartes graphiques (GPU) dédiées à la 3D

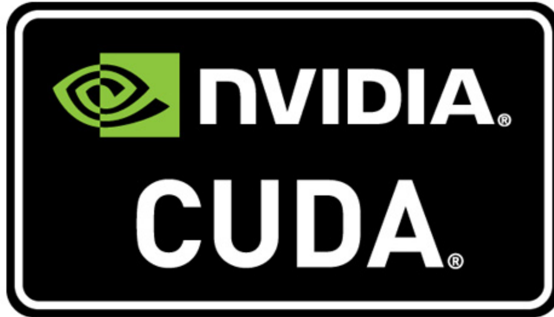
Années 2000 :

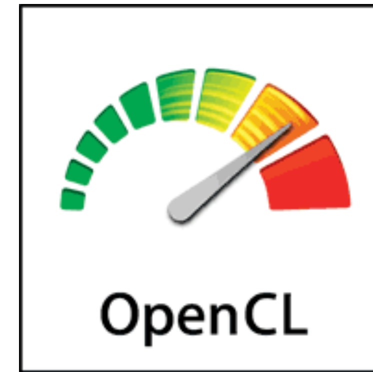


www.nvidia.com

- 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.

Après 2007 :





- CUDA devient le langage de référence pour le calcul GPGPU mais est limité aux cartes Nvidia

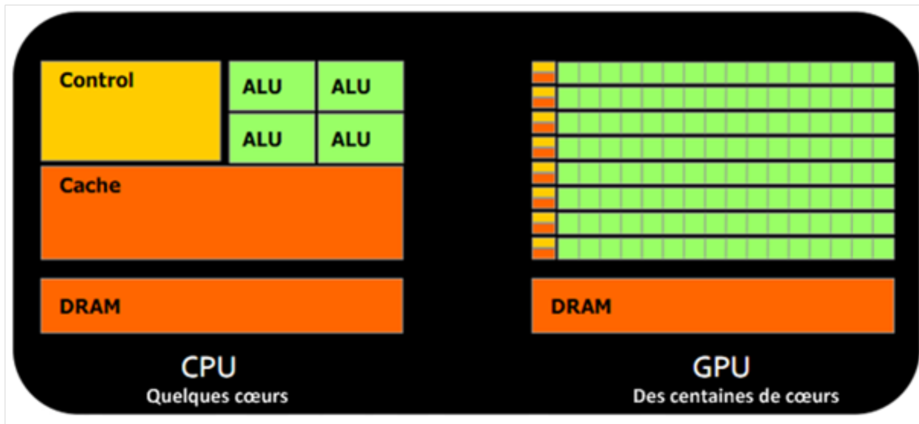
- Développement d'OpenCL, un langage ouvert pour le calcul parallèle sur GPU et CPU, par un groupe de 120 entreprises (APPLE, ATI, NVIDIA, INTEL, SUN MICROSYSTEMS, SGI, …).
  → fonctionne sur les GPU Nvidia, ATI, AMD, et les processeurs Intel
  → fonctionne sous Windows, MacOS, Linux, Android, iOS.

Ces 2-3 dernières années :

- CUDA est le leader, entre autres, pour les applications liées à l'analyse de données. L'utilisation d'OpenCL est assez courante.

- Succès récents du Deep Learning ou de XGBoost fortement liés au calcul GPGPU

CPU vs GPU



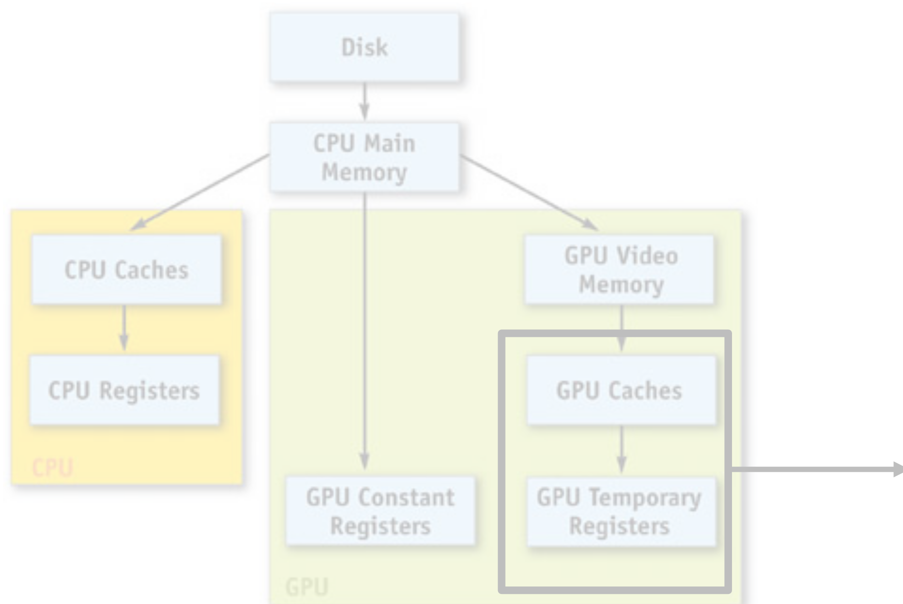
http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU

DRAM : Dynamic Random Access Memory (mémoire classique)

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

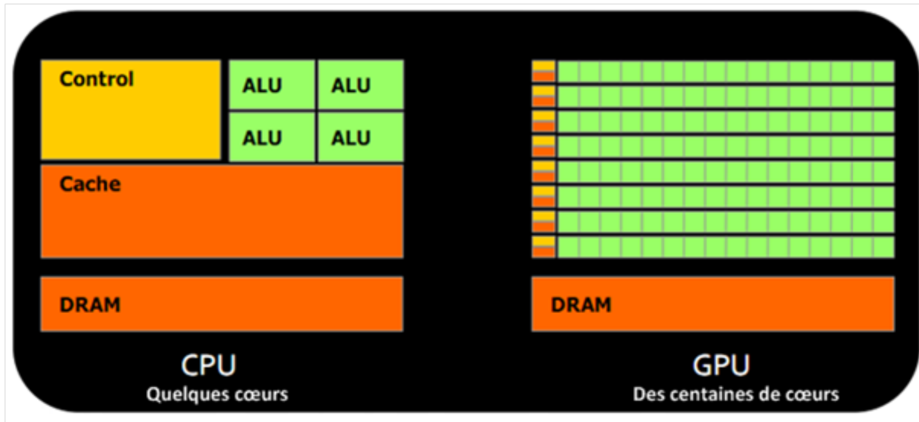
ALU : Arithmetic-Logic Unit. Effectue les calculs.

Control : Coordonne les ALU et la mémoire



- Partie largement parallélisée
- Gestion de threads et de groupes de threads
- Optimisé pour données structurées en 1D, 2D, 3D

CPU vs GPU



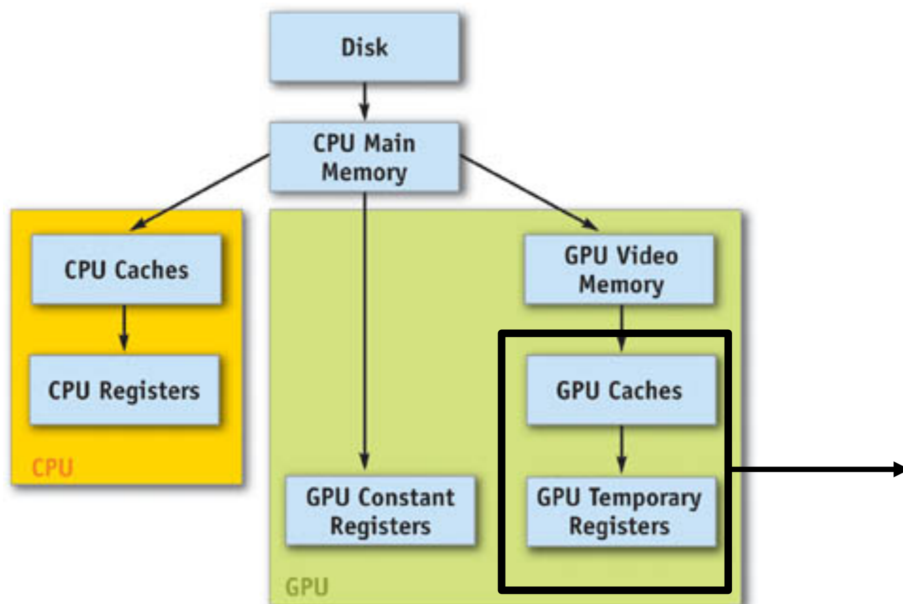
http://igm.univ-mlv.fr/~dr/XPOSE2013/GPGPU

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



- Partie largement parallélisée
- Gestion de threads et de groupes de threads
- Optimisé pour données structurées en 1D, 2D, 3D

Exemple sous Python avec l'API PyopenCL

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

Appel

```
import numpy as np
import pyopencl as cl
```

Initialisation

```
os.environ["PYOPENCL_CTX"] = '1:0'
os.environ["PYOPENCL_COMPILER_OUTPUT"] = '1'
ctx = cl.create_some_context()

queue = cl.CommandQueue(ctx)
...
```

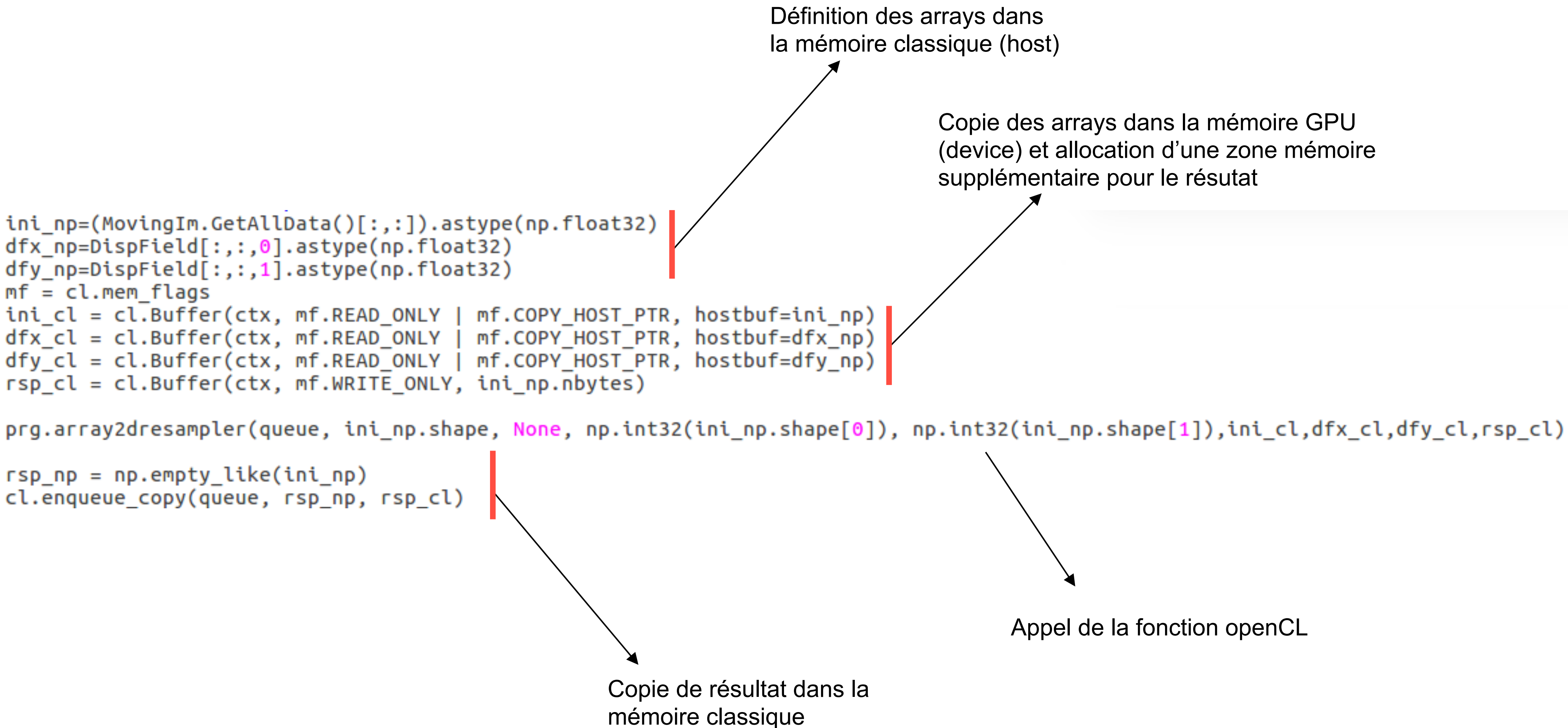
2 : Définition d'une fonction (kernel) qui s'applique en **un point** d'une matrice (array) :

```
prg = cl.Program(ctx, """
  __kernel void array2dresampler(
  const unsigned int xSize,
  const unsigned int ySize,
  __global const float *ini_cl,
  __global const float *dx_cl,
  __global const float *dy_cl,
  __global float *rsp_cl)
  {
    int rspx = get_global_id(0);
    int rspy = get_global_id(1);
    int inix = rspx+convert_int(dx_cl[rspy+ySize*rspx]+0.5);
    int iniy = rspy+convert_int(dy_cl[rspy+ySize*rspx]+0.5);
    inix=inix*(inix>=0)*(inix<xSize)+(xSize-1)*(inix>=xSize);
    iniy=iniy*(iniy>=0)*(iniy<ySize)+(ySize-1)*(iniy>=ySize);

    rsp_cl[rspy+ySize*rspx] = ini_cl[iniy+ySize*inix];
  }
""").build()
```

Position dans l'array (ici en 2D)

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



Résultats en analyse d'images médicales (stage Alexandre Martin – ENS Lyon Info)

Recalage d'images médicales 3D avec [Vialard et al., Diffeomorphic 3D Image Registration via Geodesic Shooting using an Efficient Adjoint Calculation, IJCV 2012] :

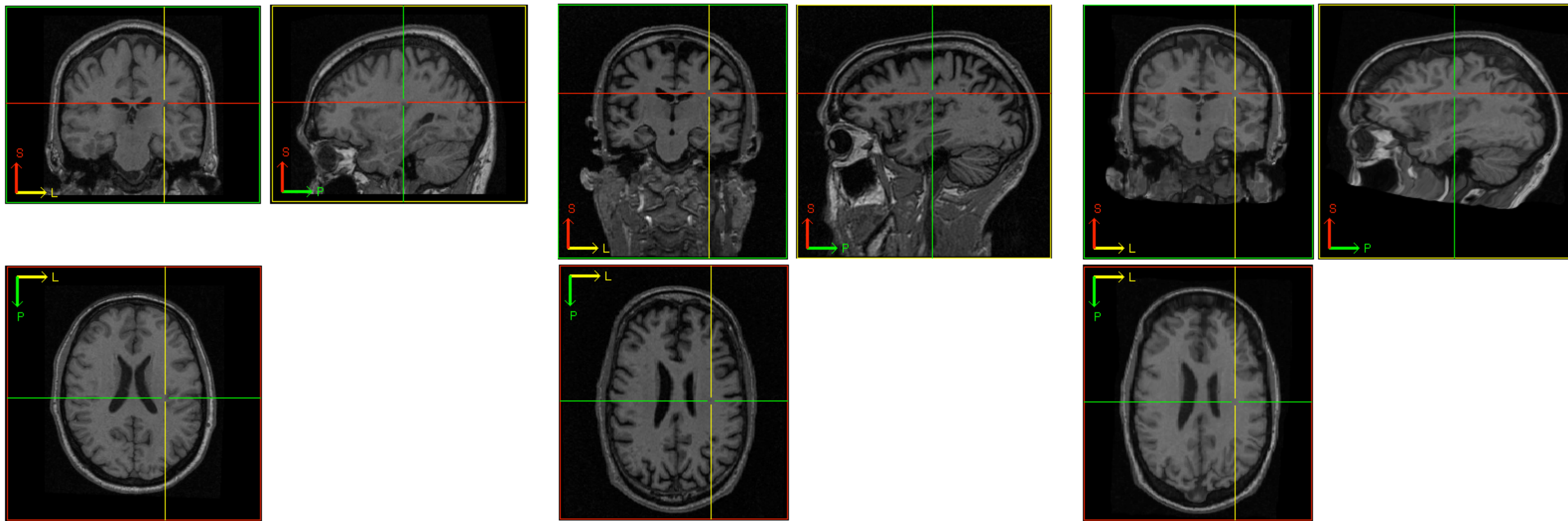


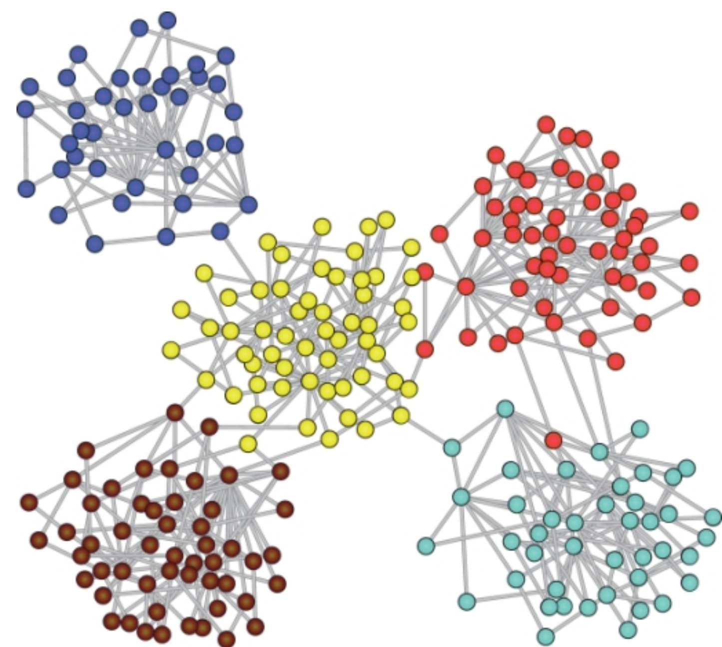
Image source **Image cible** **Image source transportée**

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

Résultats en clustering de graphes (stage Victor Bres – INPT ENSEEIHT)

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

- Graphes sociaux et routiers de grande taille
- Algorithme de croissance de régions



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

Computation time (s)	#of clusters	CPU	GPU1	GPU2	GPU3
FACEBOOK - 4×10^3 nodes	5	0.58	0.081	0.09	0.05
	10	0.12	0.080	0.08	0.06
ZBMATHS - 9×10^4 nodes	10	0.34	0.27	0.32	0.20
	100	1.50	1.26	1.17	0.88
USA_NY - 3×10^5 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 - 3×10^6 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 - 2×10^7 nodes	2	X	X	185.28	238.42
	10	X	X	213.66	X

Architectures testées :

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

Conclusion

1) Extrêmement efficace au moins pour l'imagerie et le calcul matriciel.

2) Nécessite un investissement de temps non-négligeable pour s'y mettre.

3) Efficacité dépendant de l'architecture matérielle mais aussi de la structure des données en mémoire.

4) Sans doute de nombreuses applications à venir en analyse de données