INSTITUT de MATHEMATIQUES de TOULOUSE

cnrs dépasser les frontières
www.cnrs.fr

Petit aperçu du calcul GPU et d'OpenCL

Laurent Risser

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

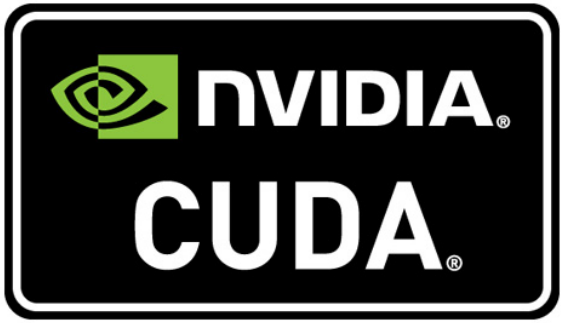
1: back in the 90’s

1) Émergence des jeux vidéos 3D

2) Gros marché pour les cartes graphiques dédiées à la 3D (GPU)

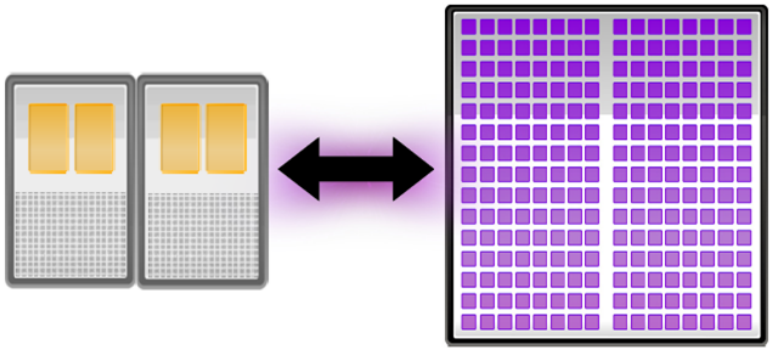
3) Baisse des coûts de solutions matérielles pour le calcul matriciel

4) Quelques geeks commencent à détourner les cartes GPU pour faire du calcul haute performance à bas coût



- Au fil des ans, CUDA devient le langage de référence pour le calcul GPU
- Limité aux cartes graphiques Nvidia
- Développement au début des années 2000 d'un langage 'libre' pour le calcul GPU (et parallele en général) par un groupe de 120 companies (ATI, NVIDIA, INTEL, SUN MICROSYSTEMS, SGI, …) : OpenCL
- Emergence peu à peu d'OpenCL car il fonctionne sur les cartes graphiques Nvidia mais aussi sur les ATI, AMD ainsi que celles intégrées dans les microprocesseurs intel récents

CPU vs GPU



Multicore CPU

Fast Serial Processing

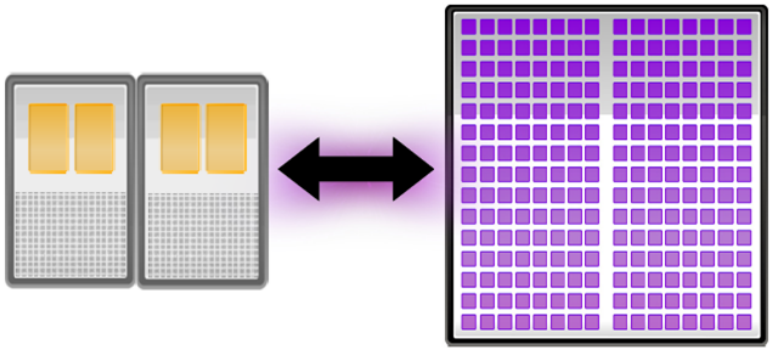
Latency-optimized cores

Manycore GPU

Scalable Parallel Processing

Throughput-optimized cores

CPU vs GPU



Multicore CPU

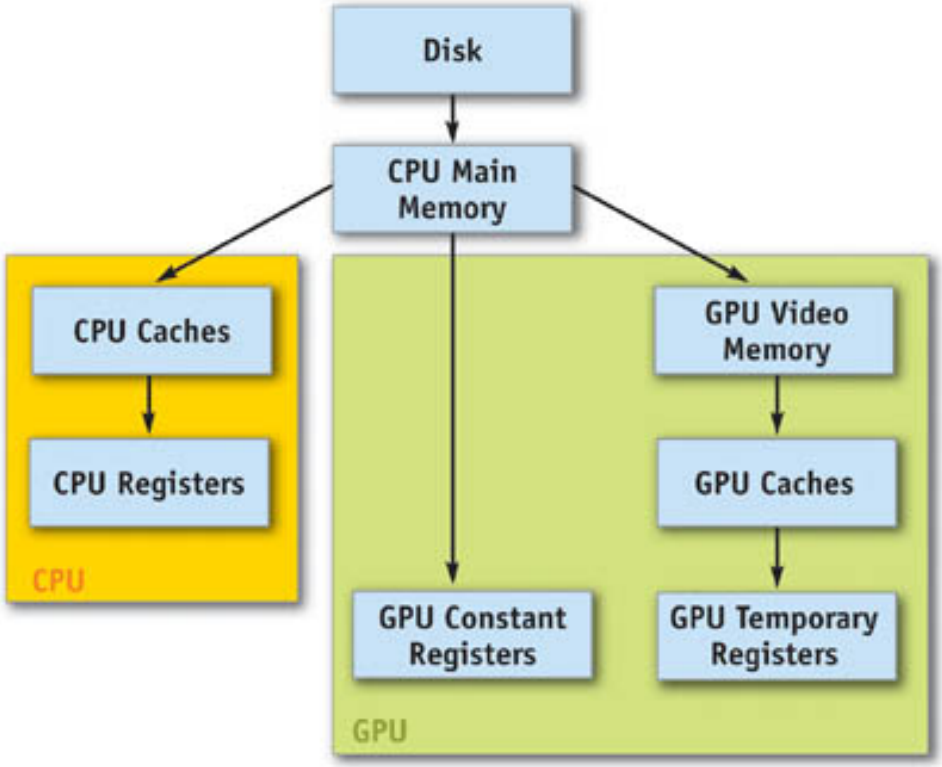
Fast Serial Processing

Latency-optimized cores

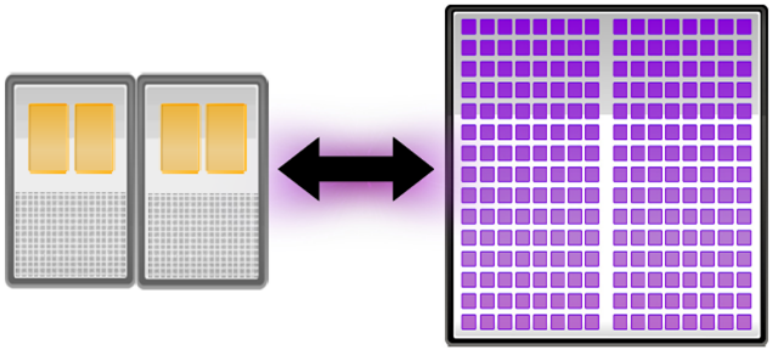
Manycore GPU

Scalable Parallel Processing

Throughput-optimized cores



CPU vs GPU



Multicore CPU

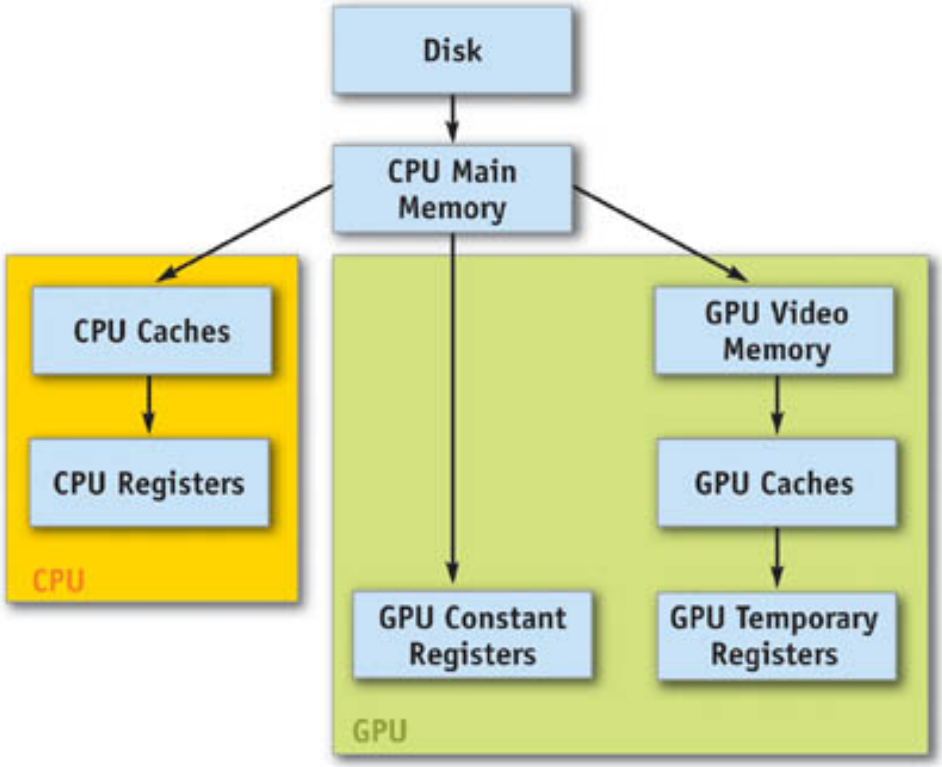
Fast Serial Processing

Latency-optimized cores

Manycore GPU

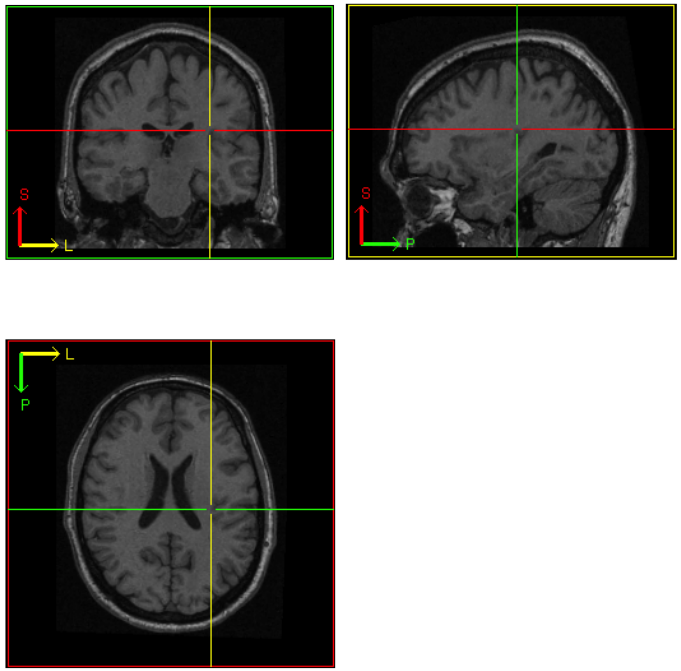
Scalable Parallel Processing

Throughput-optimized cores

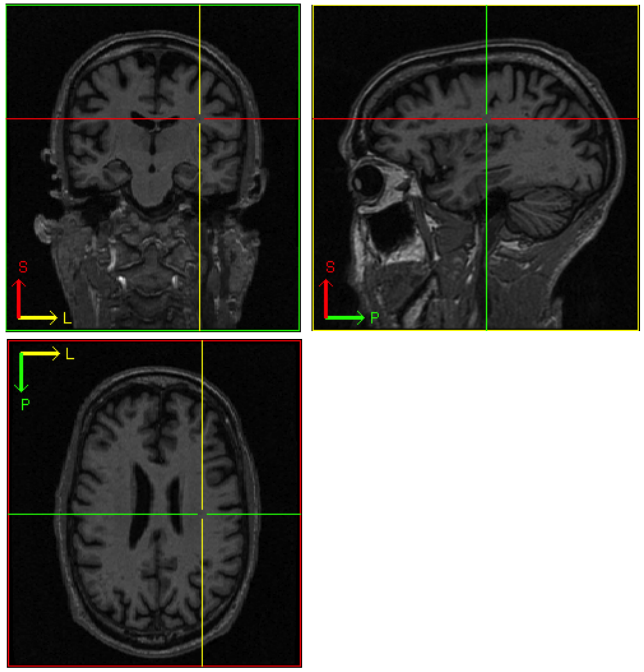


Architecture à bas coût pour faire très efficacement du calcul sur des données de taille modérée

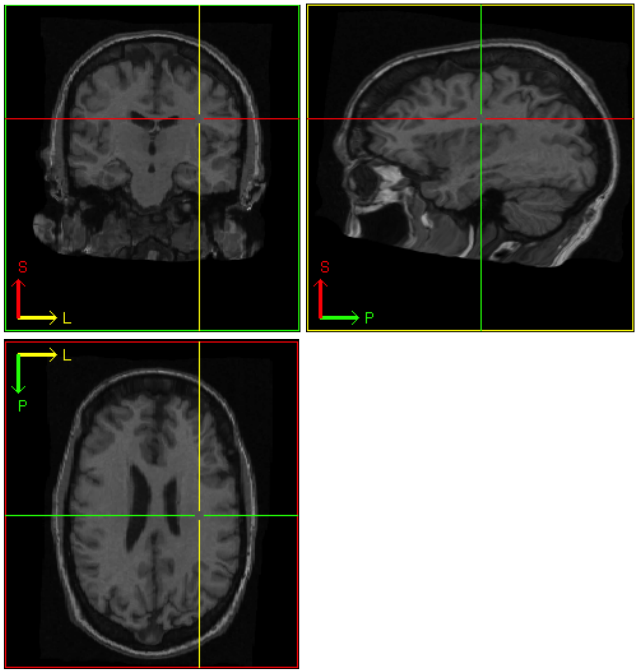
Motivations et résultats après un stage (A. Martin)



a) Source Image (256x256x198)



b) Target Image (180x256x256)



c) Registered Image (S2T)

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

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

En pratique - Installation

Installation des drivers OpenCL

→ Si le PC a un processeur Intel, les drivers se trouve là :
https://software.intel.com/en-us/articles/opencl-drivers

→ Si le PC a une carte graphique NVIDIA, les drivers se trouvent là :
http://www.nvidia.com/Download/index.aspx?lang=en-us

→ Si le PC a une carte graphique AMD, les drivers se trouvent là :
http://support.amd.com/en-us/kb-articles/Pages/OpenCL2-Driver.aspx

Identification d'un compilateur et/ou de librairies et/ou d'interface pour la programmation, par exemple :

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

→ Librairie PyOpenCL en Python : https://mathema.tician.de/software/pyopencl/

→ Interface Boost en C++ : http://www.boost.org/doc/libs/1_63_0/libs/compute

→ ...

En pratique – Exemple sous Python

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

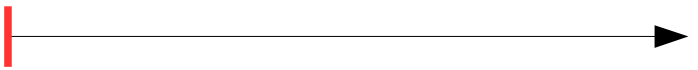
Appel

Initialisation

```
import numpy as np
import scipy
import time
import os
from xml.etree import ElementTree


#import pyopencl if installed
global usePyopenCL

try:
  import pyopencl as cl
  global usePyopenCL
  usePyopenCL=1
  os.environ["PYOPENCL_CTX"] = '1:0'
  os.environ["PYOPENCL_COMPILER_OUTPUT"] = '1'
  ctx = cl.create_some_context()
  #break
  queue = cl.CommandQueue(ctx)
except ImportError:
  usePyopenCL=0
```

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

En pratique – Exemple sous Python

(3) Appel de la fonction après avoir transféré les données en la mémoire GPU puis récupération du résultat dans la mémoire classique. La fonction est massivement parallélisée sur tous les points de l'array.

Définition des arrays dans la mémoire classique (host)

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

```
ini_np=(MovingIm.GetAllData()[:,:]).astype(np.float32)
dfx_np=DispField[:,:,0].astype(np.float32)
dfy_np=DispField[:,:,1].astype(np.float32)
mf = cl.mem_flags
ini_cl = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=ini_np)
dfx_cl = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=dfx_np)
dfy_cl = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=dfy_np)
rsp_cl = cl.Buffer(ctx, mf.WRITE_ONLY, ini_np.nbytes)

prg.array2dresampler(queue, ini_np.shape, None, np.int32(ini_np.shape[0]), np.int32(ini_np.shape[1]),ini_cl,dfx_cl,dfy_cl,rsp_cl)

rsp_np = np.empty_like(ini_np)
cl.enqueue_copy(queue, rsp_np, rsp_cl)
```

Appel de la fonction openCL

Copie de résultat dans la mémoire classique

Conclusion

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

2) … mais il faut trouver le temps de s'y mettre

3) Evaluation de l'intérêt l'été prochain pour le clusturing de graphes

4) Sans doute de nombreuses applications en analyse de données