GPU et projets académiques

Séminaire départemental

David Landry
Université Laval

24 février 2017

À propos de moi

- Étudiant à la maîtrise en Robotique Mobile
- Sous la supervision de Philippe Giguère
- Perception avec LiDAR
  - *Nuages de points*

Objectif

Susciter l’utilisation du GPGPU
(General Purpose Graphics Processing Unit)
dans les projets des étudiants du département

Résumé

(1) Motivation

(2) Qu’est-ce qu’un GPU?

(3) Comment détecter un programme parallélisable?

(4) Où commencer?

Résumé

(1) Motivation

(2) Qu’est-ce qu’un GPU?

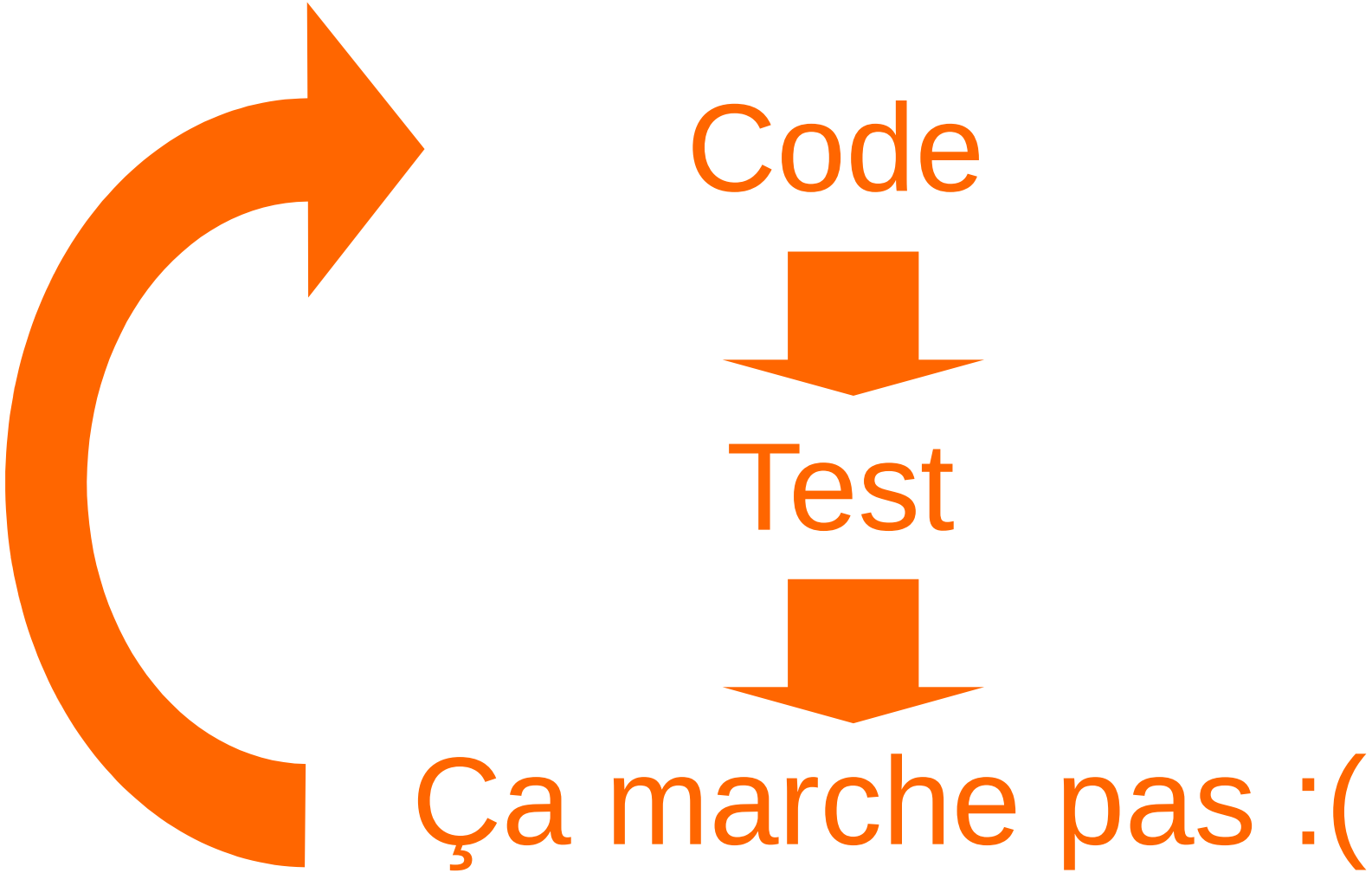
(3) Comment détecter un programme parallélisable?

(4) Où commencer?

Pourquoi s’y intéresser?

« C’est seulement un petit projet, les performances ne sont pas importantes »

Cycle de développement



Cycle de développement

- Plus le cycle est facile, plus on a envie de le répéter
- Plus on répète le cycle, meilleur est le code
- Un code plus rapide favorise l'exploration de paramètres, de nouvelles solutions
  - La confection de réseaux de neurones est un bon exemple

Pourquoi s’y intéresser?

« Je vais perdre plus de temps à apprendre à programmer des GPU que je vais en sauver »

Pourquoi s’y intéresser?

- Il existe maintenant des librairies faciles d’utilisation qui permettent une amélioration rapide du code
- On passe toujours plus de temps que prévu sur un projet
  - Le ratio temps investi/temps sauvé est donc meilleur qu’on le pense

Résumé

(1) Motivation

(2) Qu’est-ce qu’un GPU?

(3) Comment détecter un programme parallélisable?

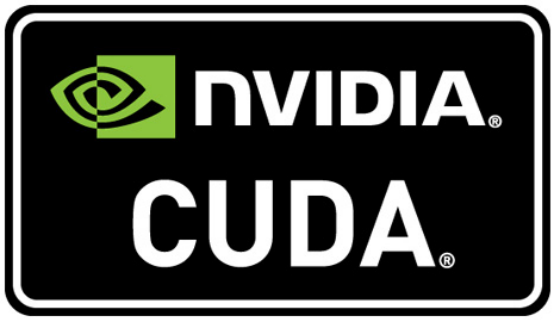
(4) Où commencer?

GPU: un historique

- Initialement créés pour accélérer le rendu graphique
  - Opérations massivement parallèles (pour chaque pixel)
  - Multiplications de matrices
- ~1998 on peut pour la première fois exécuter son propre code sur des GPU (shaders)
  - On soit encore programmer en terme de primitives graphiques (pixels, polygones...)
- ~2006 CUDA permet de programmer sur le matériel Nvidia en C
  - Plus besoin de reformuler le problème. On parle de **GPGPU**.

Interfaces vers le GPGPU

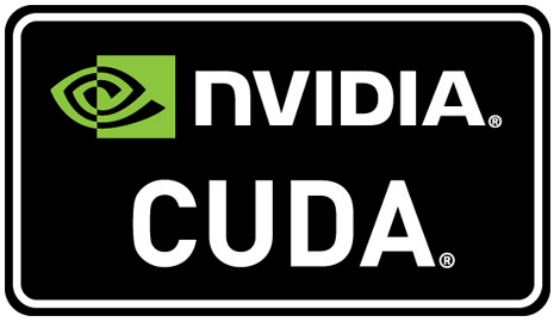
OpenCL



- OpenCL est l'API libre le plus commun pour développer sur GPU
  - Il est aussi capable de cibler des CPU multi-coeurs
- CUDA est l'api propriétaire dominant
- Dans les deux cas, on écrit du code pour GPU en C

Interfaces vers le GPGPU

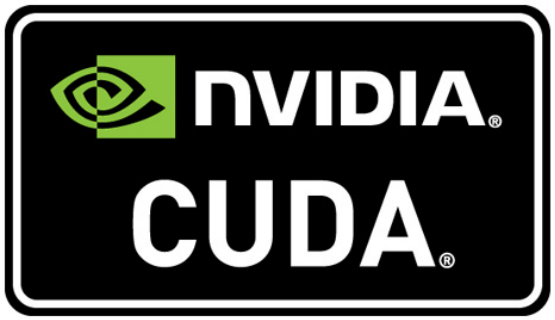
OpenCL



- Dans les deux cas, l'écriture de bon code demande des connaissances spécifiques
- Heureusement, des librairies ont été construites par dessus ces interfaces bas-niveau
- C'est sur les librairies que nous allons nous concentrer
  - Même pour utiliser les librairies, il faut être conscient du fonctionnement des GPU

Interfaces vers le GPGPU

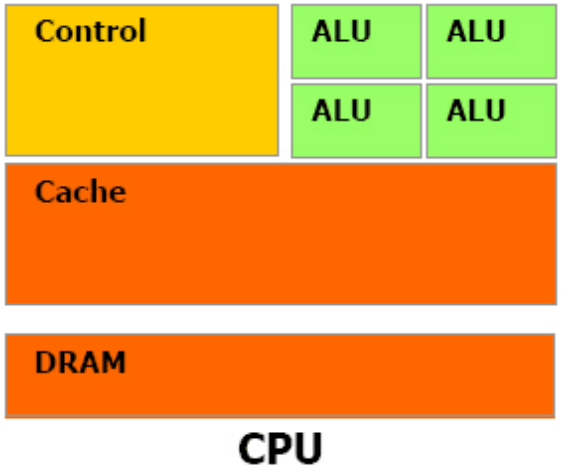
OpenCL



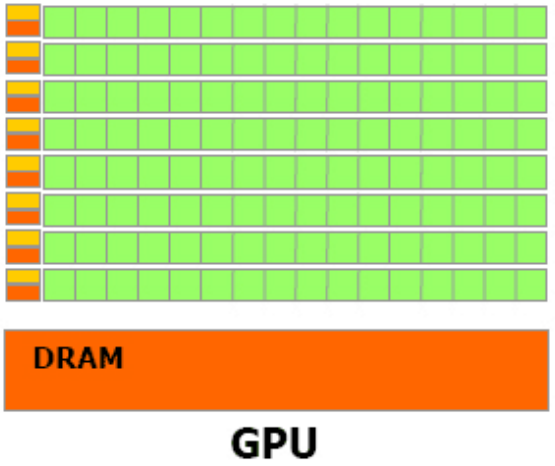
- OpenCL est supporté par une très grande variété d'accélérateurs
  - AMD Radeon
  - Nvidia
  - Intel
- CUDA est supporté seulement par les appareils Nvidia

Comparatif des architectures

Basse latence
Programmes généraux



Haut-débit
Programmes sur des données en flux



Comparatif des architectures

Basse latence

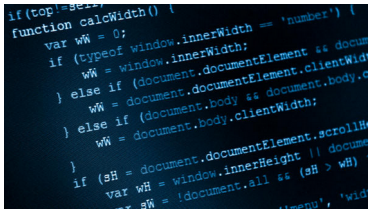
Les opérations individuelles sont rapides

Haut-débit

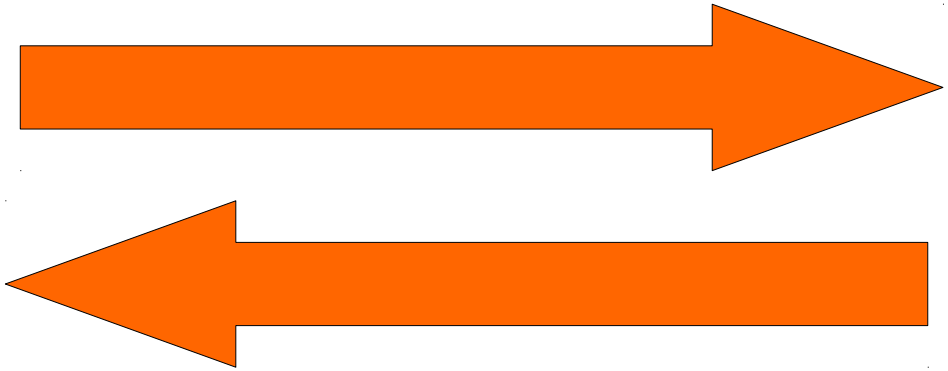
Fait plusieurs opérations plus lentement, mais en parallèle

Interactions avec GPU

Données



Kernel



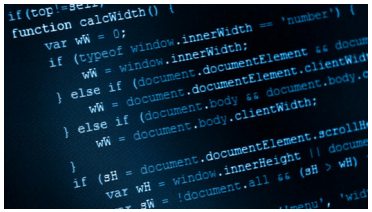
Résulat

8-32 go de RAM

24 go de RAM
(Tesla K80)

Interactions avec GPU

Données



Kernel

Résulat

8-32 go de RAM

Transferts coûteux

24 go de RAM (Tesla K80)

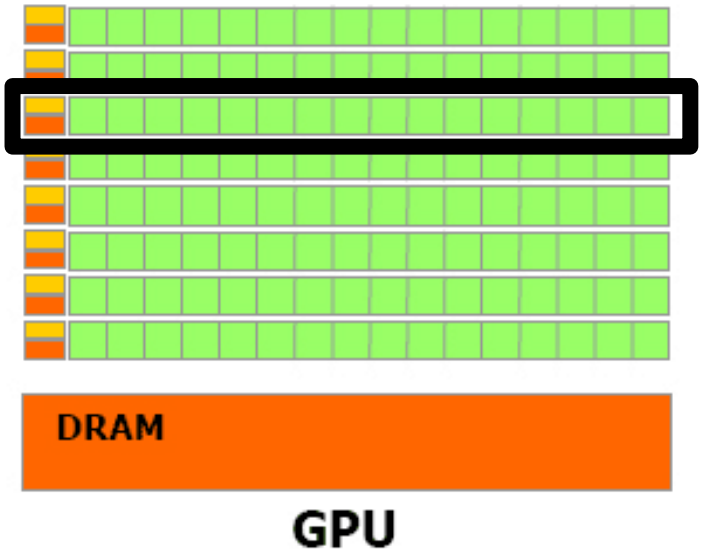
Interactions avec GPU

Morale de l’histoire

Il faut limiter les interactions entre le CPU et le GPU

À garder en tête
Branch Divergence

- Une carte graphique contient plusieurs *Streaming Multiprocessors (SM)*
  - Par exemple une GTX 1080 contient 20 SM
- Chaque SM peut exécuter entre 1-6 *warps* à la fois
  - Un *warp* est un groupe de 32 threads
- Les *warps* exécutent une seule instruction à la fois



À garder en tête
Branch Divergence

- Comme les *warps* exécutent une seule instruction à la fois, les divergences ont un impact négatif sur les performances
  - Par exemple, si un kernel contient un if-then-else, on exécute le then sur tous les threads concernés, puis le else sur tous les autres threads
  - L'exécution des deux blocs est faite séquentiellement

À garder en tête
Branch Divergence

Morale de l’histoire

Le GPU ne sera pas utilisé efficacement si votre kernel contient beaucoup des branchements

À garder en tête
Block Scheduling

- Quand on transfère les données au GPU, on les divise en **blocs**
- Les blocs doivent être indépendants entre eux et pouvoir s’exécuter dans n’importe quel ordre
- Ces conditions permettent au processeur de passer d’un bloc à l’autre sans problème

À garder en tête
Block Scheduling

- Chaque SM peut avoir jusqu'à 8 blocs en réserve
- Si un bloc est en attente de données, le SM active un autre bloc pour continuer les calculs
- C'est de cette façon qu'on dissimule la latence derrière des calculs
- Un bon nombre de blocs est donc une bonne chose

À garder en tête
Block Scheduling

Morale de l’histoire

Le GPU ne sera pas utilisé efficacement si on n’a pas suffisament de données

Résumé

(1) Motivation

(2) Qu’est-ce qu’un GPU?

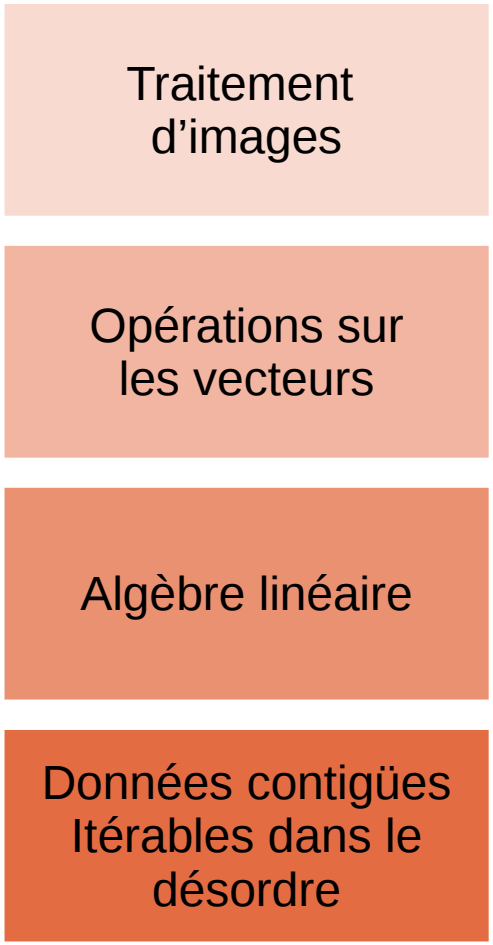
(3) Comment détecter un programme parallélisable?

(4) Où commencer?

Comment détecter les applications du GPU?

- En règle générale, on sait déjà qu'il faut
  - Éviter les interactions GPU/CPU fréquentes
  - Éviter les branchements (if)
  - Avoir une grande quantité de données à traiter
- Il faut avoir des données en *blocs traitables indépendamment*
- Ces règles abstraites sont mieux expliquées avec des exemples

Collection arbitraire et non-exhaustive des types de programmes portables sur GPU



Traitement d’images

Traitement d’images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües
Itérables dans le désordre

- Les GPUs ont été conçus pour traiter des matrices de pixels, donc c’est nécessairement un bon exemple d’utilisation
  - Filtration
  - Détection de coins

Opérations sur les vecteurs

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües
Itérables dans le désordre

- Min, max
- Sum
- Sort
- Uniq
- Dérivation numérique

Algèbre linéaire

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües Itérables dans le désordre

- Multiplications de matrices
- Systèmes d'équations linéaires
- Factorisations
  - Un des premiers programmes scientifiques à bénéficier des GPU est la factorisation LU

Données contigües et itérables dans le désordre

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües Itérables dans le désordre

- Votre code peut probablement être porté sur GPU si vos données sont...
  - Contigües en mémoire
  - Itérables dans le désordre
- Des opérations plus compliquées peuvent l'être aussi, mais c'est du cas par cas
- Parfois il faut travailler le problème de façon à le ramener sous une forme parallélisable

Résumé

(1) Motivation

(2) Qu’est-ce qu’un GPU?

(3) Comment détecter un programme parallélisable?

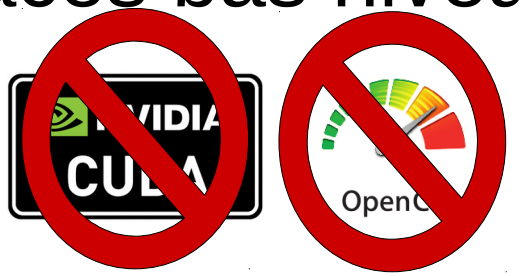
(4) Où commencer?

Le matériel

- Les GPU Nvidia sont « dans le vent »
- Il existe un *GPU Grant Program* (professeurs seulement) http://bit.ly/gpugrant
- Un portable avec GPU peut faire une bonne partie du chemin
- Le super-ordinateur Helios, de Calcul Québec, a des GPU sur tous les noeuds
- Dans le pire des cas, on en trouve pour 100-1000$ sur le marché, selon les performances désirées

Interface de programmation

- On cherche à éviter les interfaces bas niveau comme CUDA et OpenCL



  - C’est difficile à déboguer
  - Il faut être très familier avec l’architecture pour écrire du code performant
  - Le code sera plus difficilement portable à d’autres cibles d’exécution
- On va plutôt préférer utiliser des abstractions qui utilisent le GPU pour nous

Choisir ses outils

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües Itérables dans le désordre

Choisir ses outils

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües Itérables dans le désordre

OpenCV

« OpenCV est une bibliothèque graphique libre spécialisée dans le traitement d'images en temps réel »

- Licence: BSD
- Disponible depuis le code source et sur Ubuntu
- Utilisable avec Python, C++, Java

OpenCV

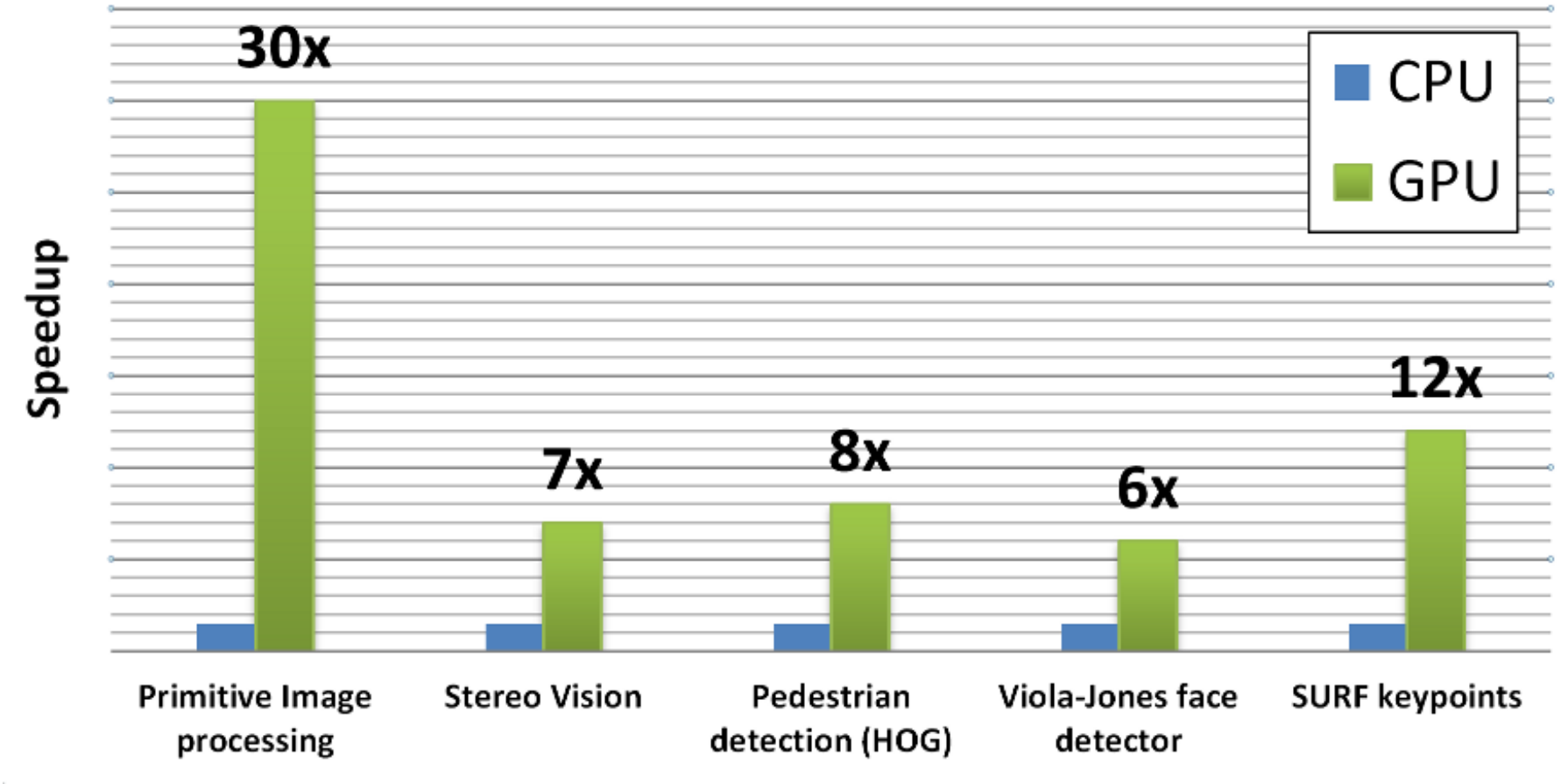
Pour accélérer les calculs de OpenCV avec GPU il faut

- Installer CUDA
- Faire un *build* depuis le code source avec quelques flags Cmake (ex. -D WITH_CUDA=ON)
- Indiquer à OpenCV de faire les transferts CPU/GPU dans les programmes

OpenCV

Visite d’un code d’exemple

Gains de performance



Choisir ses outils

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües Itérables dans le désordre

{} ArrayFire

- Librairie multi-fonctions qui peut s’exécuter sur plusieurs cibles
  - CUDA (GPU Nvidia)
  - OpenCL (Presque tous les accélérateurs)
  - CPU (si nécessaire)
- Utilisable en C++, Python, Rust, Node.js
  - Compatible avec Numpy
- Licence: BSD

ArrayFire

Une visite sur la documentation de la librairie indique qu'elle a des modules pour des domaines variés

- Algorithmes sur des vecteurs
- Algèbre linéaire
- Traitement de signal
- Statistiques

{} ArrayFire

Pour accélérer ses calculs avec ArrayFire il faut

- Installer CUDA
- Se procurer ArrayFire (depuis les paquets Ubuntu ou depuis le code source)
- [Optionnel] Installer l’interface Python

{ArrayFire}

Démonstration

ArrayFire

Exemples de gains

Opération	Speedup Nvidia GT630	Speedup Nvidia Tesla

Choisir ses outils

Traitement d'images

Opérations sur les vecteurs

Algèbre linéaire

Données contigües
Itérables dans le désordre

OpenACC

Directives for Accelerators

« The OpenACC Interface describes a collection of compiler directives to specify loops and regions of code [...] to be offloaded from a host CPU to an attached accelerator. »

- Licence: Propriétaire
- Utilisable avec C, C++ (et Fortran)

OpenACC

Directives for Accelerators

- OpenACC est un ensemble de directives qui indiquent au compilateur quelles sections du code sont parallélisables
- De là, le compilateur fait de son mieux pour porter le code sur GPU
  - Encore une fois, le code qu'on écrit n'est pas spécifique à notre cible d'exécution
- La code est portable sur des accélérateurs Nvidia, AMD, et Intel

OpenACC
Directives for Accelerators

```
#pragma acc kernels
{

for (int i=0; i < N; i++) {
    C[i] = A[i] + B[i];
}

}
```

Ceci indique au compilateur de

1) Identifier le parallélisme dans le code
2) Identifier les données à transférer sur GPU
3) Créer un *kernel*
4) Faire exécuter le calcul sur GPU

OpenACC
Directives for Accelerators

Pour la programmation GPU par directives, un compétiteur de OpenACC est OpenMP

OpenACC Directives for Accelerators	OpenMP
Propriétaire	Open source
Principalement descriptif	Principalement prescriptif

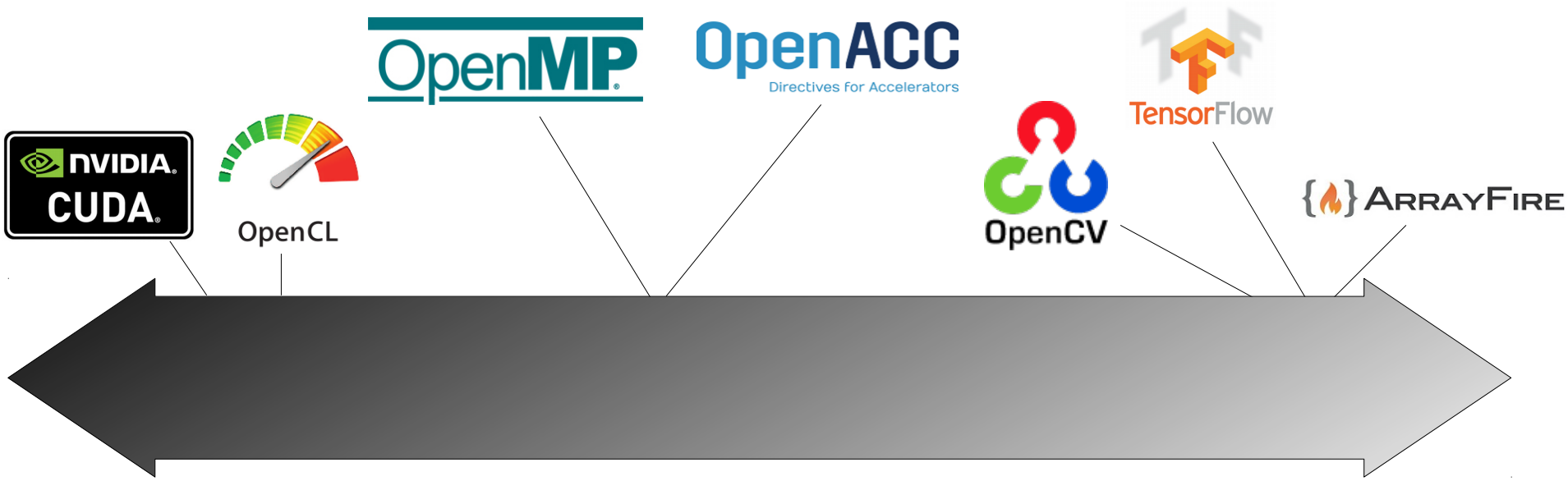
OpenACC
Directives for Accelerators

- OpenACC est un standard implémenté sur plusieurs compilateurs
- Le support par le compilateur GCC n’est pas encore au point
- Il faut donc se procurer le compilateur propriétaire PGI
  - La *Community Edition* est disponible gratuitement

OpenACC
Directives for Accelerators

Visite d’un code d’exemple

Synthèse – Librairies GPU



Expérience GPU requise

Synthèse – Librairies GPU

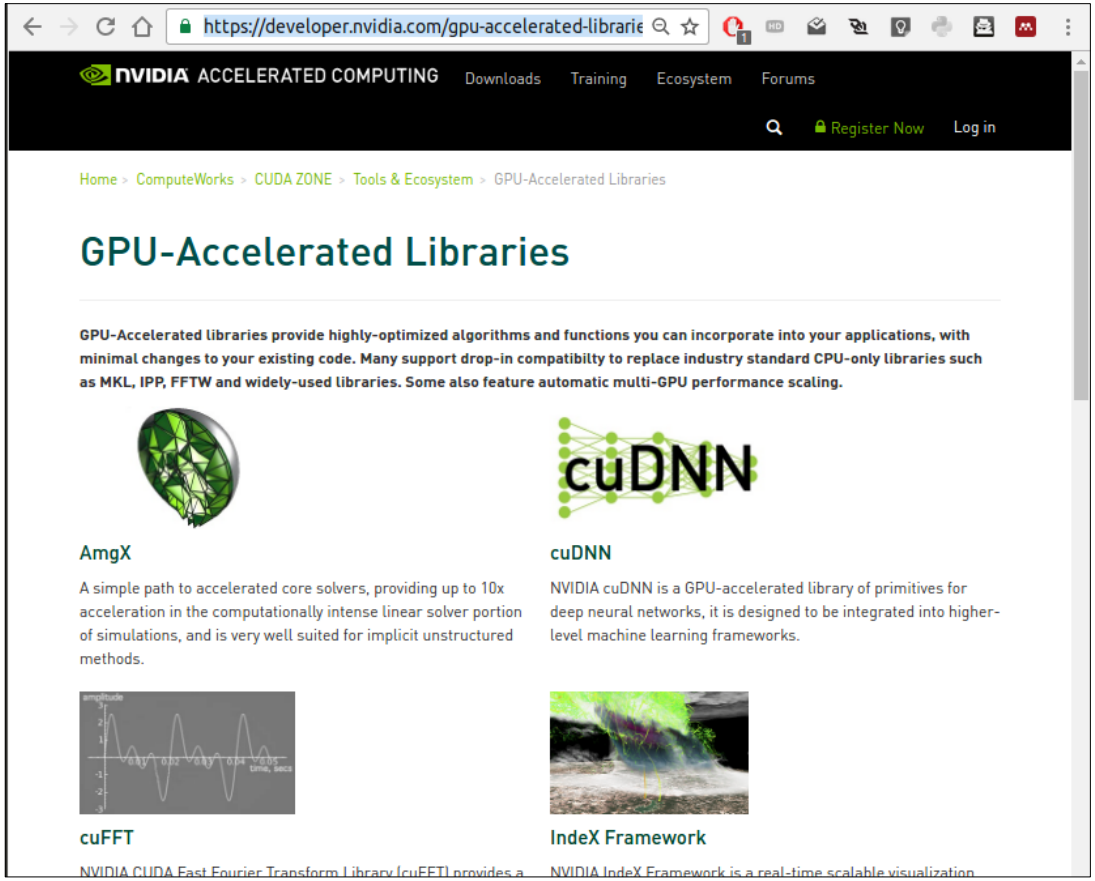
Librairie	C/C++	Java	Python
	X	X	X
	X		X
	X		

Ressources importantes

Prochaines formations

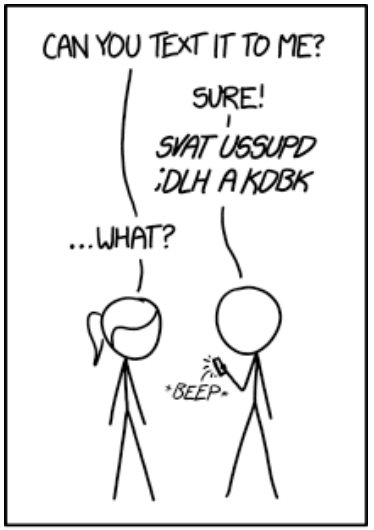
http://bit.ly/formations2

Ressources importantes



https://developer.nvidia.com/gpu-accelerated-libraries

Merci! Des questions?



Merci à Marc Parizeau pour les conseils

La librairie Thrust

« Thrust is a parallel algorithms library which resembles the C++ Standard Template Library (STL). »

- Licence: Apache
- Disponible gratuitement sur internet

La librairie Thrust

- Essentiellement, elle réimplémente certaines opérations de la STL sur GPU
- Des algorithmes tels que max, sum, sort, partition... sont donc disponibles facilement
- On peut aussi coder ses propres kernels avec des fonctions telles que transform ou reduce (moins recommandé)

Comment se la procurer

- Elle est fournie avec le toolkit CUDA de Nvidia
- Il faut donc la télécharger et l'installer à http://bit.ly/get-cuda
- Si l'installation est réussie, Thrust va s'utiliser comme une librairie C++ normale.

Exemple

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <algorithm>
#include <cstdlib>

int main(void)
{
  // generate random data serially
  thrust::host_vector<int> h_vec(100);
  std::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device and compute sum
  thrust::device_vector<int> d_vec = h_vec;
  int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
  return 0;
}
```