UF I5AISE51
Dimensionnement et évaluation des architectures
Introduction à la programmation massivement parallèle

—

Partie 1
Introduction à la programmation parallèle

—

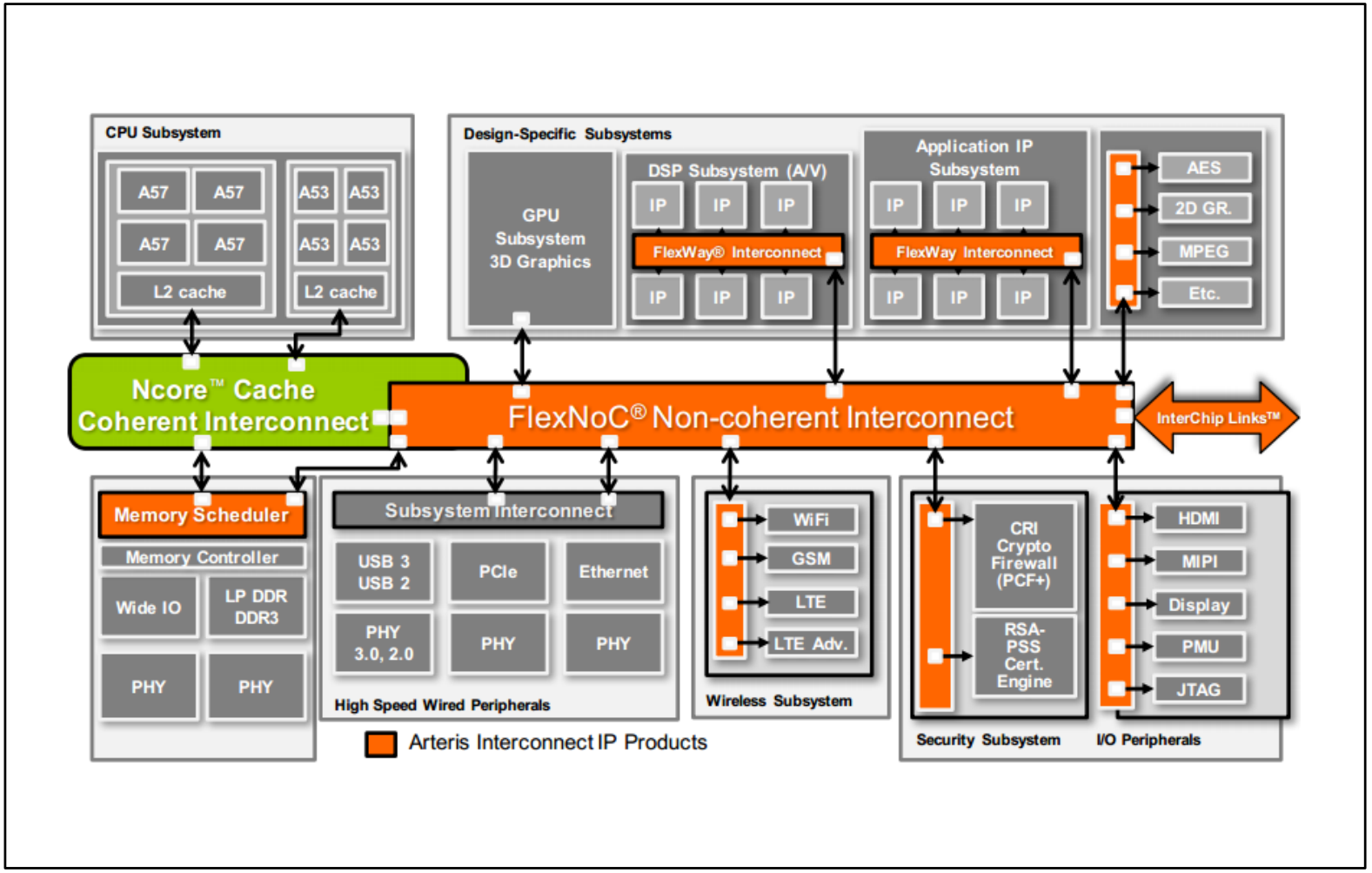
P.-E. Hladik

INSA Toulouse

4 janvier 2021

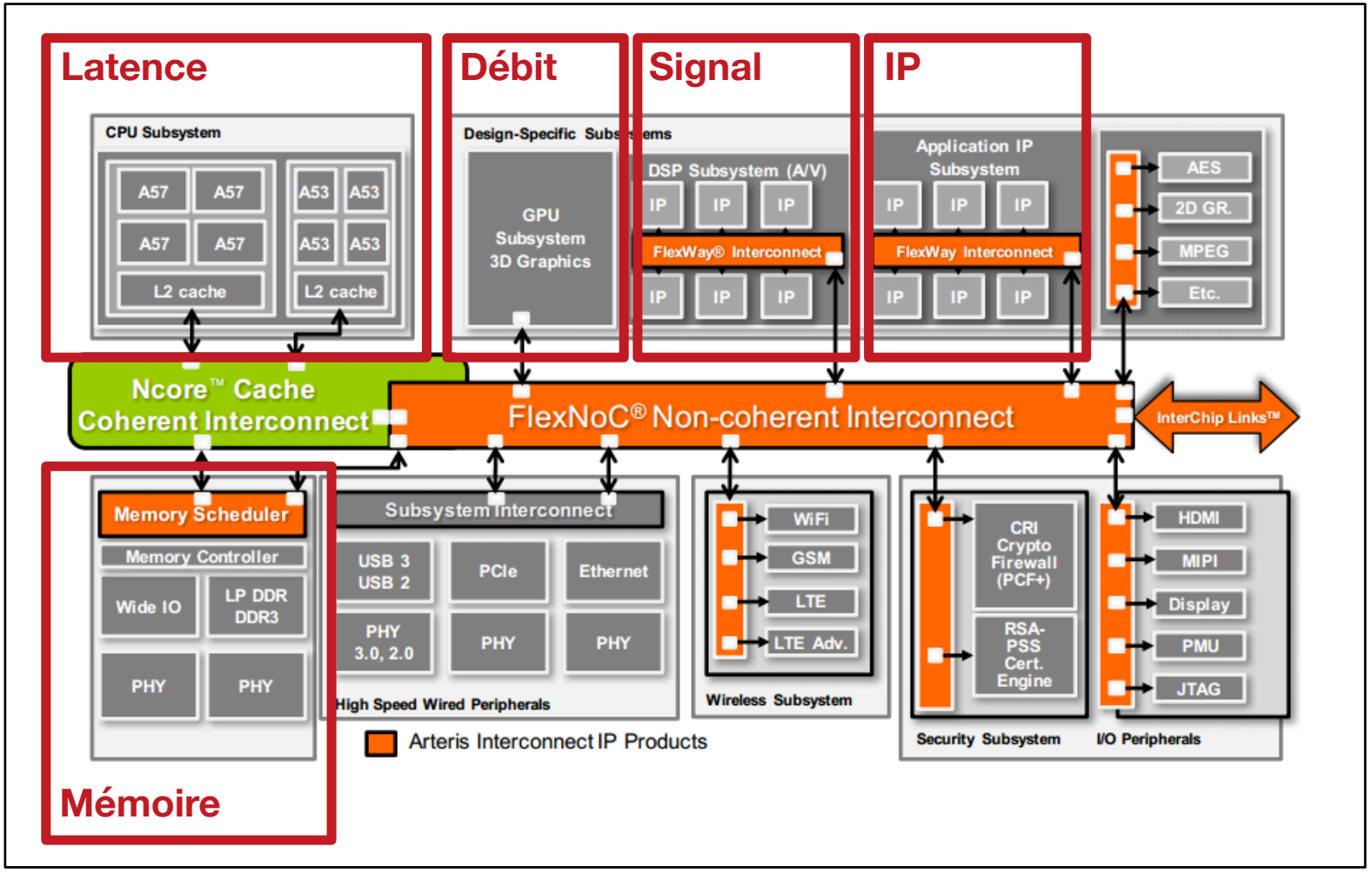
Plan

Exemple d'architecture hétérogène parallèle



https://www.arteris.com/hubfs/2016-05-24-arteris-ncore-overview-PDF-FINAL.pdf

Exemple d'architecture hétérogène parallèle

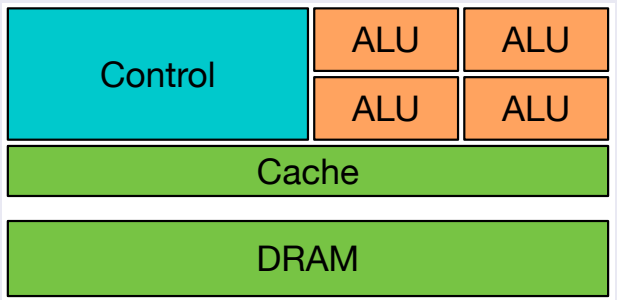


https://www.arteris.com/hubfs/2016-05-24-arteris-ncore-overview-PDF-FINAL.pdf

Architecture CPU vs GPU

CPU : Latency Oriented Design (latence)

Optimisé pour les performances d'un code séquentiel.

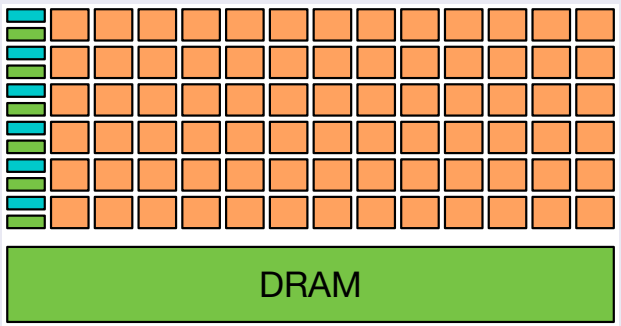


- ALU puissante : réduction de la latence des opérations
- Larges caches : conversion des accès longs à la mémoire en des accès courts au cache (réduction de la latence)
- Contrôle d'exécution sophistiqués : prédiction de branchement, pipeline, data forwarding, etc. pour réduire la latence

Architecture CPU vs GPU

GPU : Throughput Oriented Design (débit)

Optimisé pour les performances d'un code parallèle (augmente le débit, c.-à-d. le travail réalisé par unité de temps)



- Petites caches : accélération du débit mémoire
- Contrôle simple : pas de prédiction de branchement, pas de data forwarding
- ALU : nombreuses unités, latence importante mais de nombreux pipeline pour un grand débit
- Grand nombre de threads matériels

Architecture CPU vs GPU

CPU

- pour les parties séquentielles où la latence est importante
- peut être 10x plus rapide qu'un GPU pour du code séquentiel

GPU

- pour les parties parallèles avec un haut débit souhaité
- peut être 10x plus rapide qu'un CPU pour le code parallèle

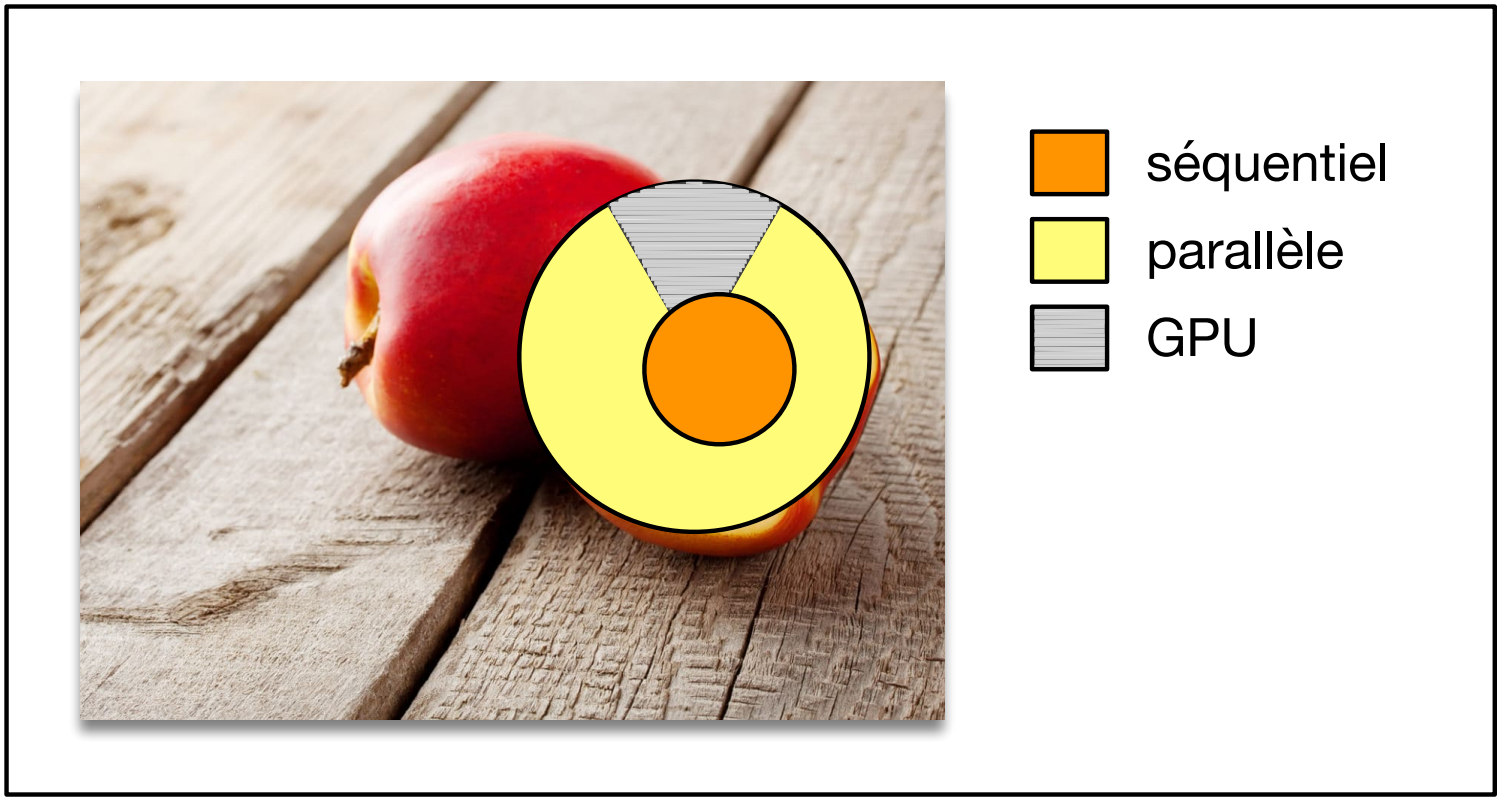
The winner is...

De bonnes performances pour une application s'obtient en utilisant judicieusement des CPU et des GPU.

Augmenter la vitesse des application

Exemple tiré de Programming Massively Parallel Processors

Augmenter la vitesse des application



Exemple tiré de Programming Massively Parallel Processors

Mais c'est quoi un traitement parallèle ?

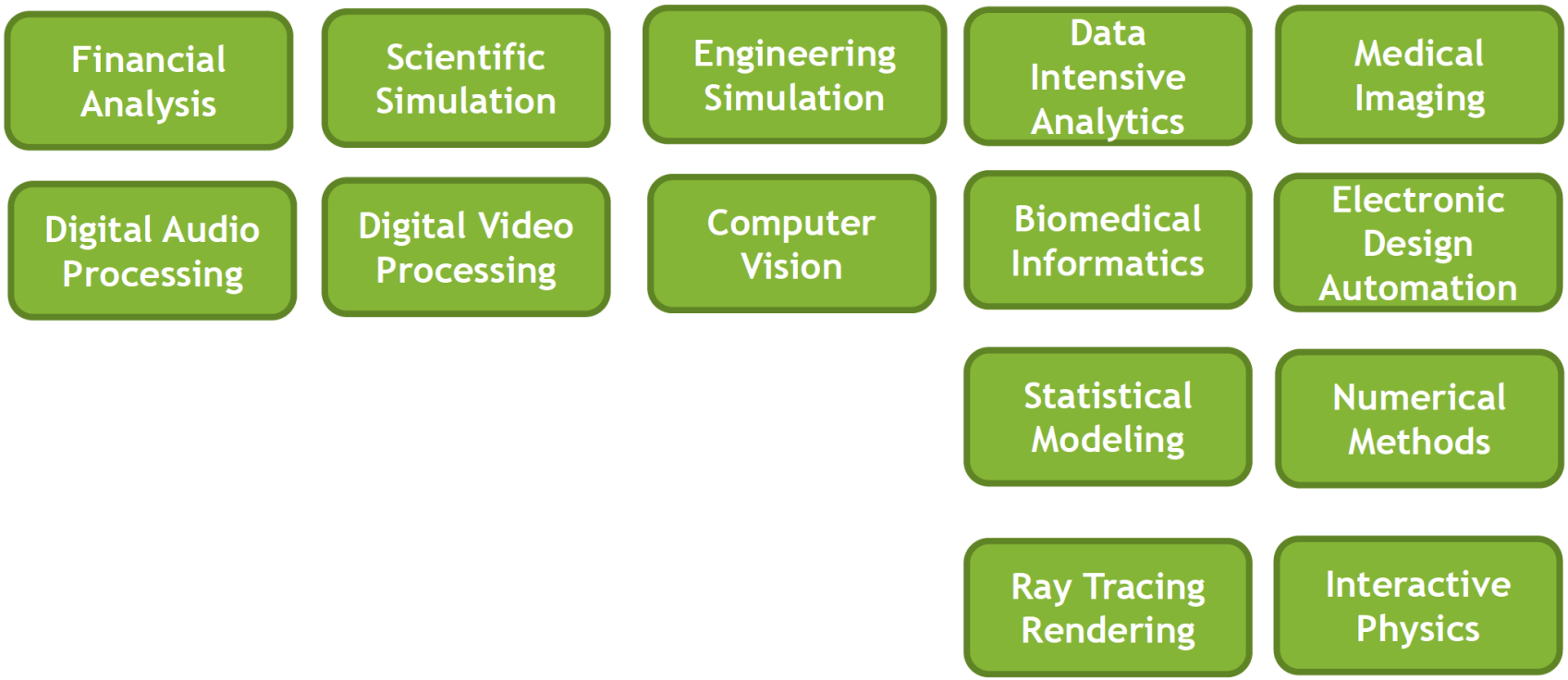
Parallélisme de tâches

- décomposition de l'application en tâches
- les traitements des tâches sont indépendants

Parallélisme des données

- traitement des données en parallèle
- les traitements sont identiques

Domaines d'application du calcul parallèle hétérogène



[Lecture 1.2 Heterogenous computing, GPU Teaching Kit Nvidia]

La programmation de GPU

Trois façon de développer des applications sur GPU

Utilisation de bibliothèques

- Facile à utiliser
- Performant
- Qualité

Directives de compilation

- Facile à utiliser
- Code portable
- Perf. incertaines

Langage de programmation

- Performant
- Flexible
- Verbeux

La programmation de GPU

Trois façon de développer des applications sur GPU

Utilisation de bibliothèques

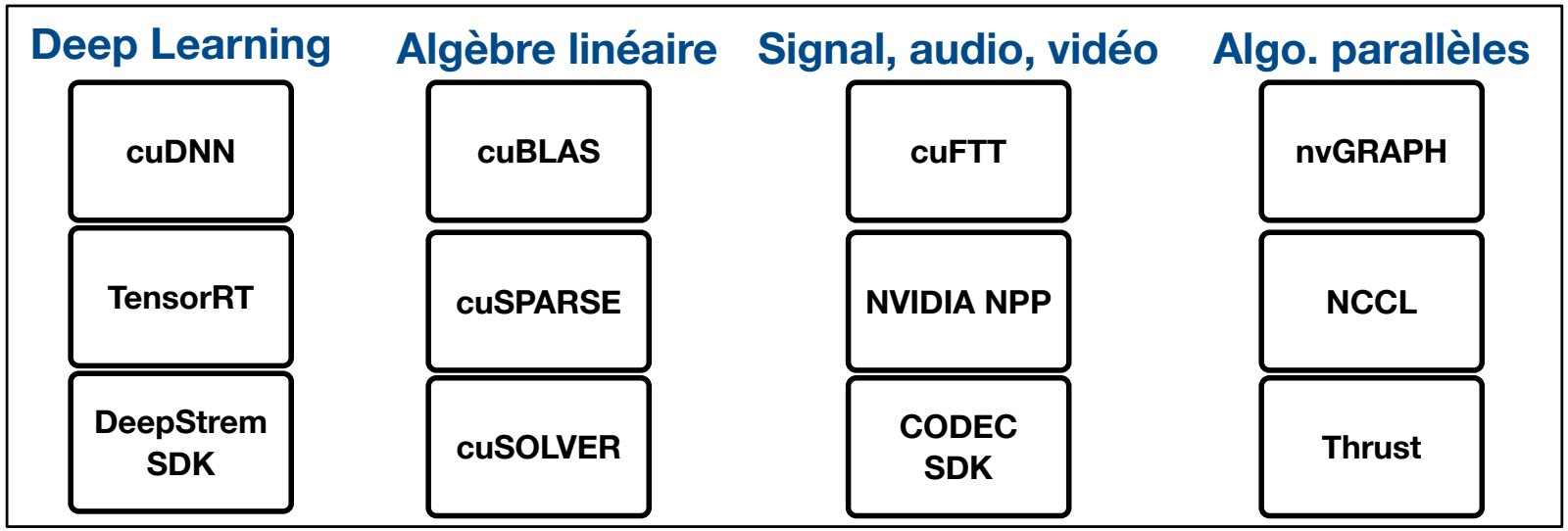
- Facile à utiliser
- Performant
- Qualité

Directives de compilation

- Facile à utiliser
- Code portable
- Perf. incertaines

Langage de programmation

- Performant
- Flexible
- Verbeux



La programmation de GPU

Trois façon de développer des applications sur GPU

Utilisation de bibliothèques

- Facile à utiliser
- Performant
- Qualité

Directives de compilation

- Facile à utiliser
- Code portable
- Perf. incertaines

Langage de programmation

- Performant
- Flexible
- Verbeux

OpenACC : directives de compilation C, C++ et FORTRAN

```
#pragma acc parallel loop
copyin(input1[0:inputLength],input2[0:inputLength]),
copyout(output[0:inputLength])
for(i = 0; i < inputLength; ++i) {
  output[i] = input1[i] + input2[i];
}
```

La programmation de GPU

Trois façon de développer des applications sur GPU

Utilisation de bibliothèques

- Facile à utiliser
- Performant
- Qualité

Directives de compilation

- Facile à utiliser
- Code portable
- Perf. incertaines

Langage de programmation

- Performant
- Flexible
- Verbeux

Analyse numérique ▶	MATLAB, Mathematica, LabVIEW
Python ▶	PyCUDA, Numba
FORTRAN ▶	CUDA Fortran, OpenACC
C ▶	CUDA C, OpenACC
C++ ▶	CUDA C++, Thrust
C# ▶	Hydridizer

La programmation de GPU

Trois façon de développer des applications sur GPU

Utilisation de bibliothèques

- Facile à utiliser
- Performant
- Qualité

Directives de compilation

- Facile à utiliser
- Code portable
- Perf. incertaines

Langage de programmation

- Performant
- Flexible
- Verbeux

CUDA-C

- NVIDIA fournit un compilateur CUDA-C : nvcc
- nvcc compile le code pour le GPU puis le transmet au compilateur hôte (par exemple g++)
- Peut être utilisé pour compiler et éditer les liens des applications réservées à l'hôte (i.e. CPU)
- Outils de débug : Nsight, CUDA-GDB, CUDA MEMCHECK)
- Outils de profilage de performance : Nsight, NVVP, NVPROF, NVTX

Un peu de pratique avec le TD1

Objectifs

- compiler avec `nvcc` et exécuter un code sur un GPU
- récupérer et afficher les informations du *device*