

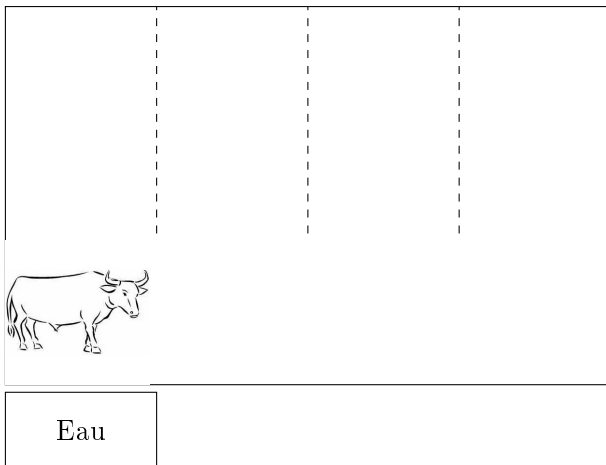
Introduction aux GPUs

Amina Guermouche

ENSEIRB-MATMÉCA

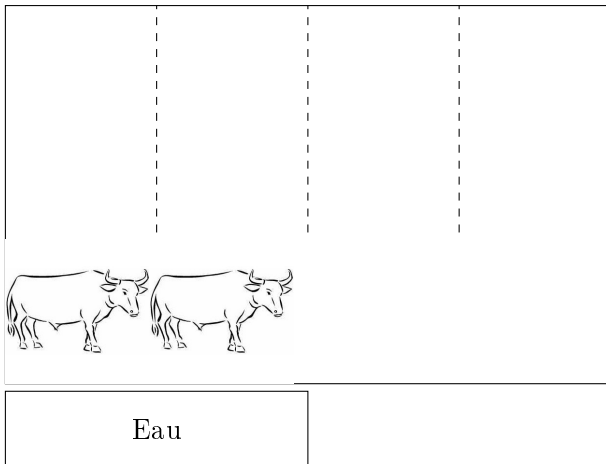
23 Novembre 2022

Un peu d'histoire



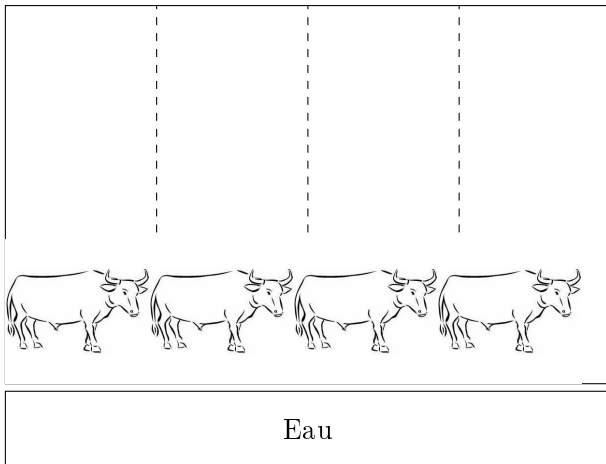
Un peu d'histoire

- Augmentation du nombre de CPU
- 😊 Plus rapide
- 😞 Plus coûteux



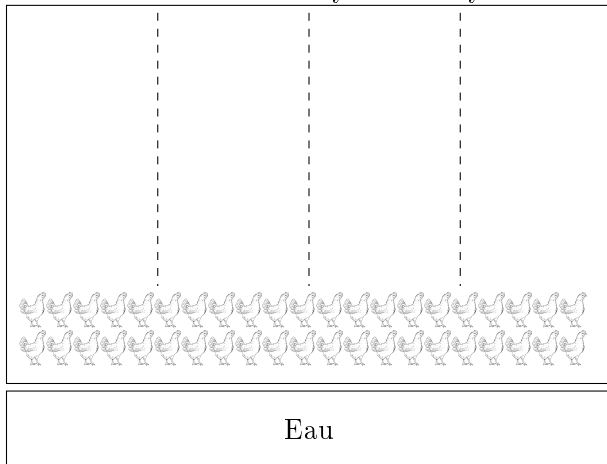
Un peu d'histoire

- Augmentation du nombre de CPU
- 😊 Plus rapide
- 😞 Plus coûteux



Un peu d'histoire

"If you were plowing a field, which would you rather use : Two strong oxen or 1024 chickens ?" Seymour Cray



Plan

- ① GPU VS CPU
- ② Architecture des GPUs modernes
- ③ Programmation des GPUs

GPU VS CPU

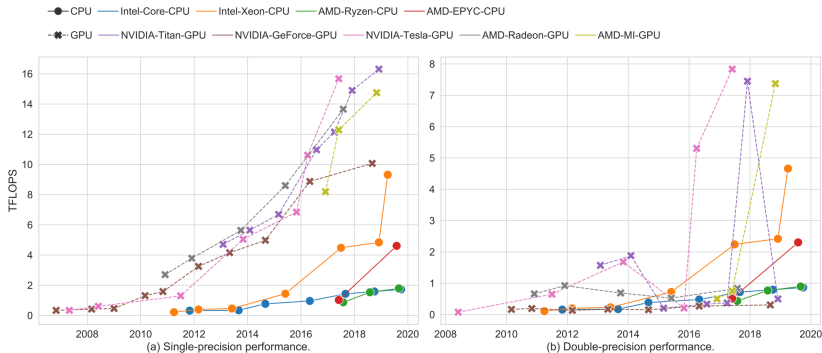


Fig. 6. Comparing single-precision and double-precision performance of CPUs and GPUs.

Référence : Summarizing CPU and GPU Design Trends with Product Data,
<https://arxiv.org/abs/1911.11313>

GPU VS CPU

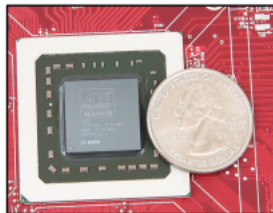


Intel i7 Quad-Core

~ 100 GFLOPS Peak

730 millions de transistors

4 threads + SSE vector
instructions



AMD Radeon HD 587

~ 2.7 TFLOPS Peak

2.2 milliards de transistors

CPU VS GPU : Démonstration

<https://www.youtube.com/watch?v=-P28LKWTzrI>

Pourquoi une telle différence

- Latence VS throuput
- Parallélisme de tâche VS parallélisme de données
- Multi-thread VS SIMD
- Des dizaines de threads VS des dizaines de milliers de threads

Latence et throughput

- La latence est le délai entre le moment où une opération est initiée, et le moment où ses effets deviennent détectable
 - Une voiture a une latence plus faible qu'un bus (plus rapide)
- Throughput (débit) est la quantité de travail effectué sur une durée
 - Un bus a un throughput plus élevé qu'une voiture (plus de personnes à la fois)

Latence et throughput

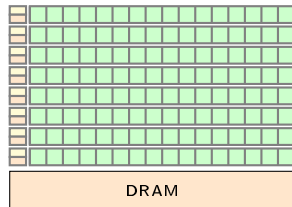
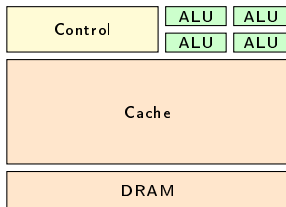
- Les CPU doivent **minimiser la latence** (négligeant le throughput 😞)
 - Un input du clavier
 - Nécessité d'utiliser des caches
 - Les CPUs maximisent les opérations en dehors du cache (pre-fetch, exécution out-of-order, ...)
- Les CPU ont besoin de caches de grande taille

Latence et throughput

- Les CPU doivent **minimiser la latence** (négligeant le throughput 😞)
 - Un input du clavier
 - Nécessité d'utiliser des caches
 - Les CPUs maximisent les opérations en dehors du cache (pre-fetch, exécution out-of-order, ...)
- Les CPU ont besoin de caches de grande taille
- Les GPU sont des processeurs à **latence et throughput élevés**
 - Ils n'ont pas besoin de large cache
 - Plus de transistors peuvent être dédiés au calcul

Pourquoi une telle différence de performance?

- Plus de transistors sont dédiés au traitement des données au lieu de la gestion des caches
- Chip de même taille mais avec plus d'ALU, donc plus de threads pour le calcul



Gestion des threads sur le GPU

- **Comment faire**
 - Synchronisation entre autant de threads (comment l'éviter)
 - Ordonnancement, commutation de contexte
 - Programmation
- Les threads sur les GPUs sont :
 - Indépendants (pas de synchronisation)
 - SIMD (coût d'ordonnancement réduit)
 - Programmation par block de threads

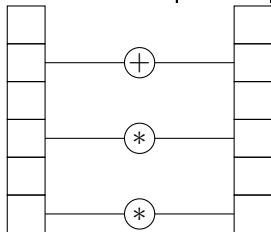
Gestion des threads sur le GPU

- **Comment faire**
 - Synchronisation entre autant de threads (comment l'éviter)
 - Ordonnancement, commutation de contexte
 - Programmation
- Les threads sur les GPUs sont :
 - Indépendants (pas de synchronisation)
 - SIMD (coût d'ordonnancement réduit)
 - Programmation par block de threads
- Applications *data parallel*
- Applications graphiques, traitement d'images, physique, informatique, ...
- Plus il y a de données, plus les GPUs sont efficaces

Pallélisme CPU VS GPU

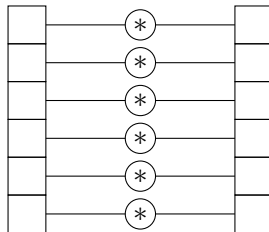
CPU : Parallélisme de tâches

- Exécution simultanées de plusieurs fonctions sur différents cœurs et sur des données identiques ou pas



GPU : Parallélisme de données

- Exécution simultanée de la même fonction par plusieurs cœurs sur différentes données



Stream processing

- L'unité fondamentale d'un GPU est le *"stream processor"*
 - Un grand ensemble de données (*"stream"*)
 - Exécuter les mêmes opérations (*"kernel"* ou *"shader"*) sur toutes les données
- Plusieurs optimisations pour améliorer le throughput
 - Mémoire et cache local on-chip pour réduire le nombre d'accès à la mémoire externe
 - Les threads sont groupés pour de meilleurs accès mémoire
 - Réduire la latence et le *"stall"*

Pourquoi les GPU ?

- Grâce aux jeux vidéos, les GPU sont :
 - très populaires
 - massivement produits (le coût est ainsi réduit)
 - Les GPUs tolèrent une large latence
- Moins de cache
- Plus de place pour les unités de calcul
- Plus de threads
 - Les GPU sont massivement parallèles :
 - Opérations (translations, rotations, ...) sur les pixels peuvent être réalisées en parallèles
 - Beaucoup d'unités de calcul
 - Mémoire locale de grande taille
 - Large bande passante mémoire

Plan

- 1 GPU VS CPU
- 2 Architecture des GPUs modernes
- 3 Programmation des GPUs

Vue générale du GPU Tesla P100 (P pour Pascal)

GPU P100 :

- 56 *Stream Process (SM)* de 32 cœurs
- 64 CUDA cores par SM
 - 1 CUDA core = 1 opération flottante par clock pour un thread
- 16GB de DRAM
- Cache L2 de 4096KB
- Connexion avec CPU via PCI express et NVLink
- Connexion entre GPU avec NVLink

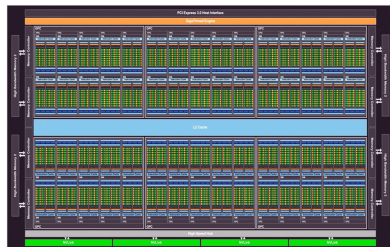


Figure – Pascal GP100
(pascal-architecture-whitepaper.pdf)

Vue détaillée du GPU Tesla P100

- 64 cœurs simple précision
- 32 cœurs double précision
- 16 unités Load/Store (LD/ST) par SM
- 16 *Special Function Units* (SFU)
 - sin, cosin, racine carrée, ...
- 64KB de mémoire partagée par SM



Référence : NVIDIA Tesla P100 white paper

Pour aller plus loin : GPU Tesla V100 (V pour Volta)

- 84 SM
- 32 cœurs double précision par SM
- Tensor cores (pour la phase d'entraînement en machine learning)

Référence : [NVIDIA Tesla V100 white paper](#)

Plan

- ① GPU VS CPU
- ② Architecture des GPUs modernes
- ③ Programmation des GPUs

Programmation GPU

Application d'un kernel, écrit comme un code séquentiel, à plusieurs données

- **CUDA**
 - C, C++, Fortran, Python
- **OpenCL**
 - Supporté par différent constructeurs (AMD et Nvidia)
 - Modèle d'exécution parallèle similaire à CUDA (mais une terminologie différente)
 - Quelques différences présentées ici :
https://people.eecs.berkeley.edu/~demmel/cs267_Spr13/Lectures/CatanzaroIntroToGPUs.pdf
- **OpenAcc**

Différents constructeurs

- NVidia (CUDA)
- AMD (Hip qui fonctionne aussi pour les GPU NVidia et a une syntaxe très proche de CUDA ce qui facilite le portage du code)
- Intel Ponte Vecchio (PVC) (pas beaucoup d'informations sur comment les programmer)

Plan du cours

- 1 Programmation avec CUDA
- 2 Gestion de la mémoire
- 3 Mesure de la consommation d'énergie
- 4 Introduction à OpenACC

Bibliographie

- CUDA C Programming
John Cheng, Max Grossman, Ty McKerchre, 2014
- C for CUDA by example
Jason Senders et Edoirt Kandrot, Addison-Wesley, 2011
- Programming Massively Parallel Processors
David B. KIRK et Wen-mei W. HWU, Morgan Kaufmann, 2010
- The CUDA handbook
Nicholas WILT, Addison-Wesley, 2013
- <http://developer.nvidia.com/cuda/nvidia-gpucomputing-documentation>