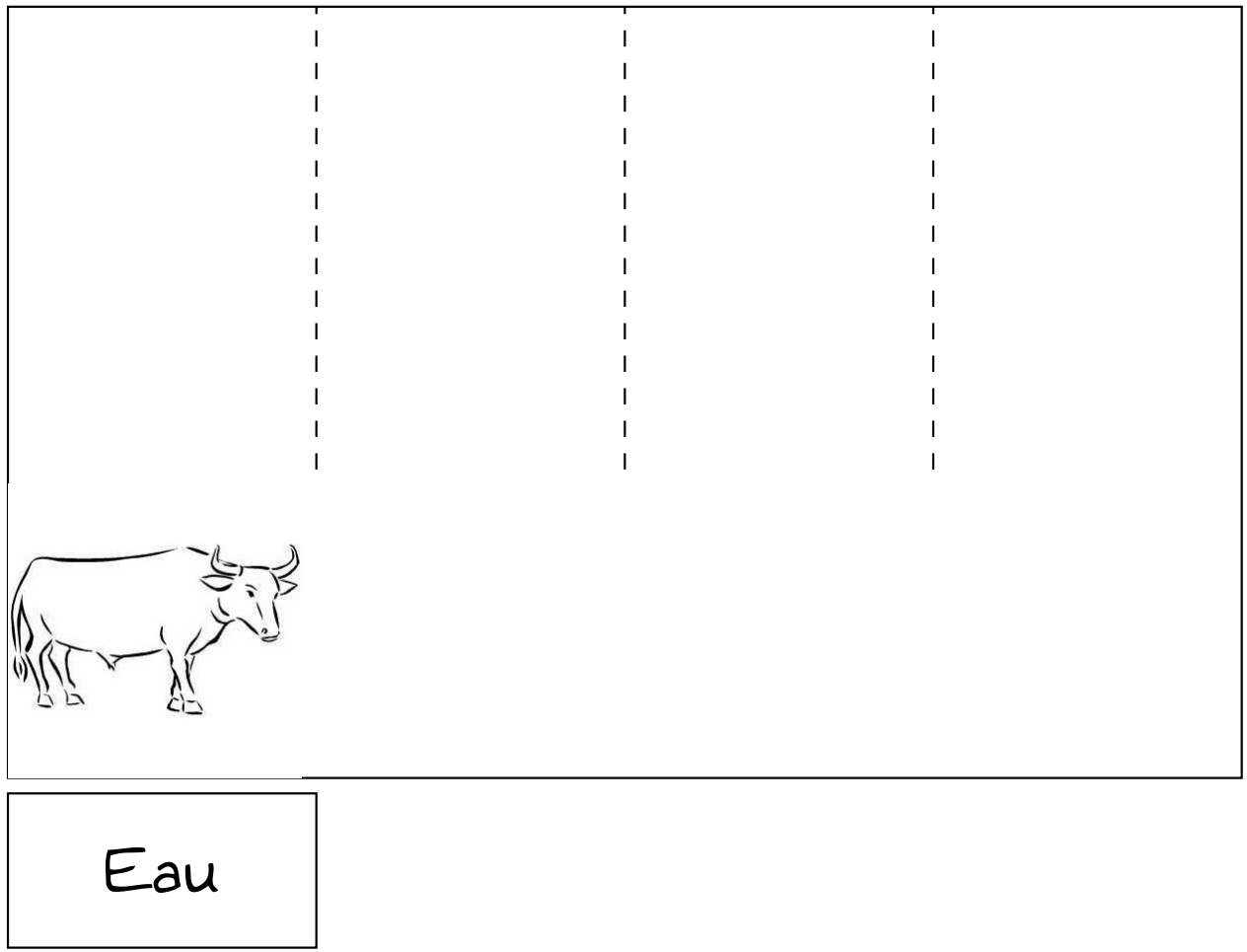
Introduction aux GPGPUs

Amina Guermouche

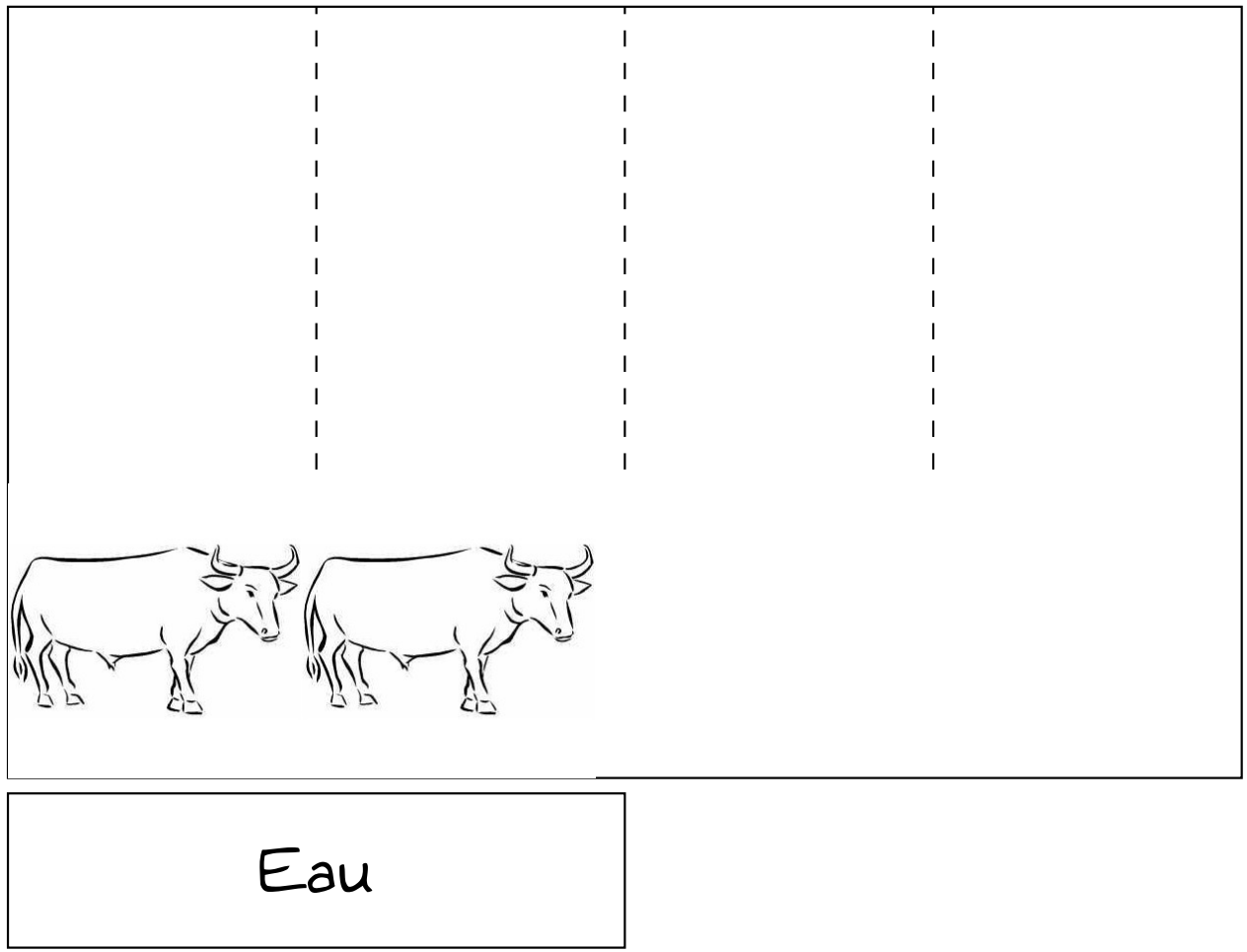
Télécom SudParis

Un peu d'histoire



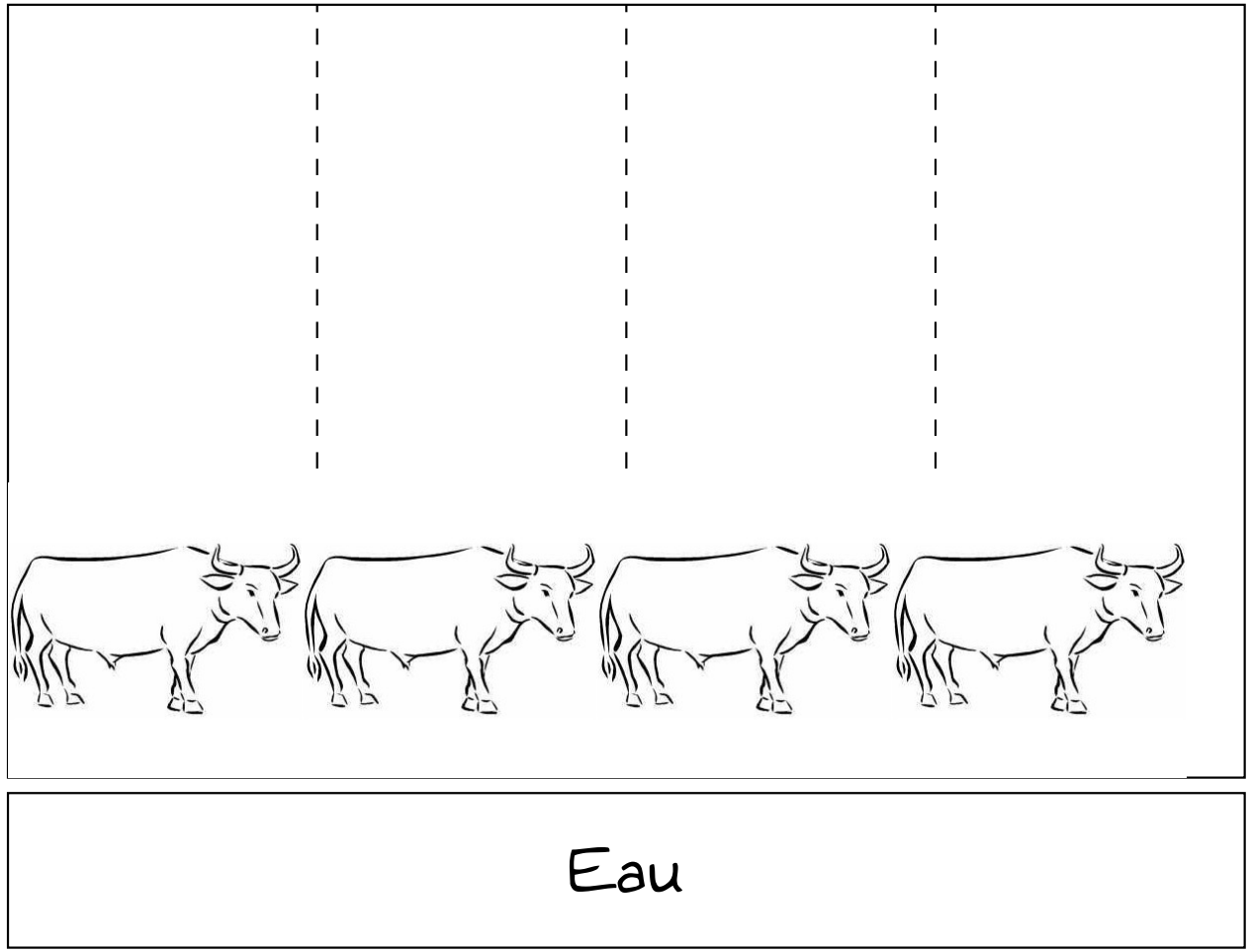
Un peu d'histoire

- Augmentation du nombre de CPU
- ☺ Plus rapide
- ☹ Plus coûteux



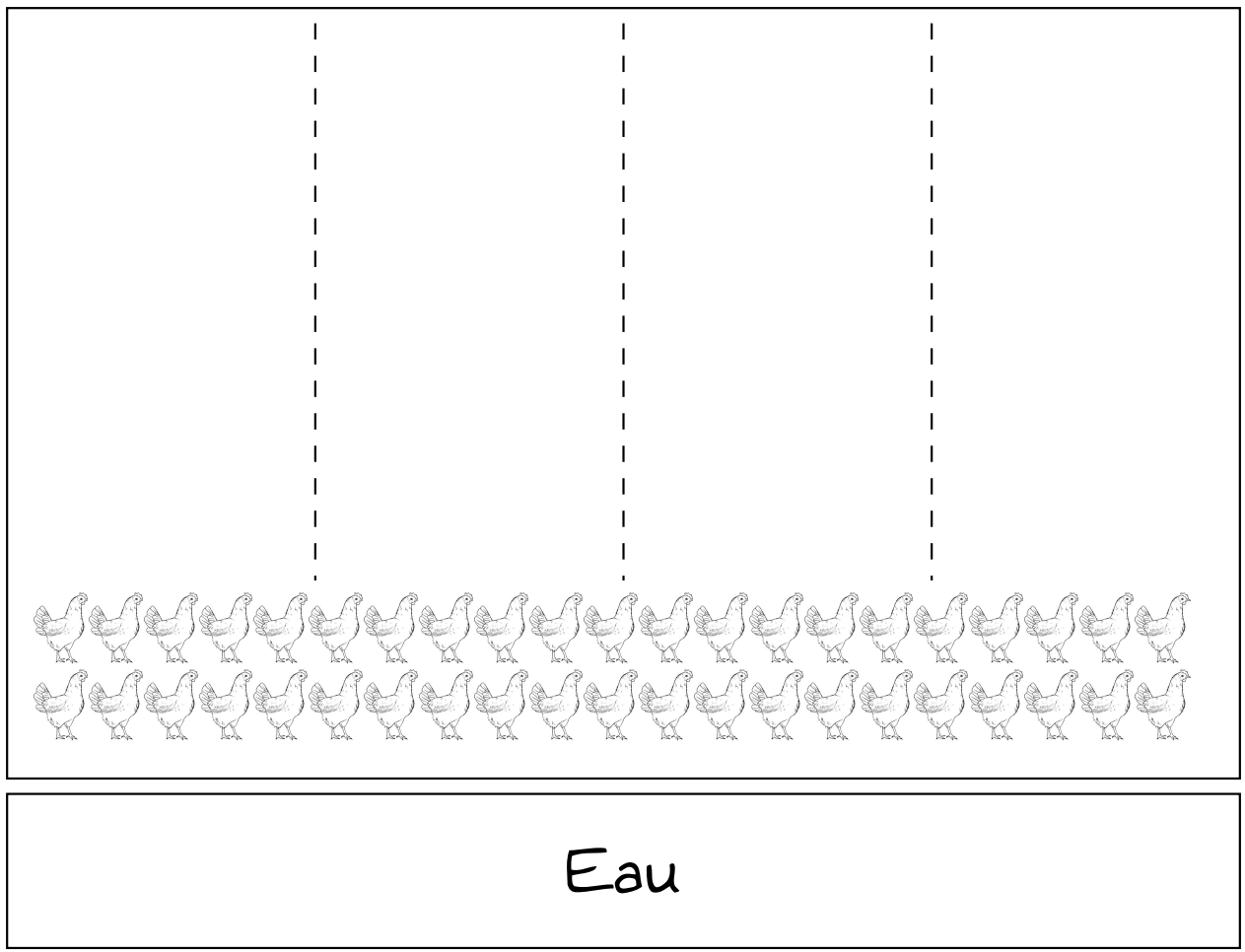
Un peu d'histoire

- Augmentation du nombre de CPU
- ☺ Plus rapide
- ☹ Plus coûteux



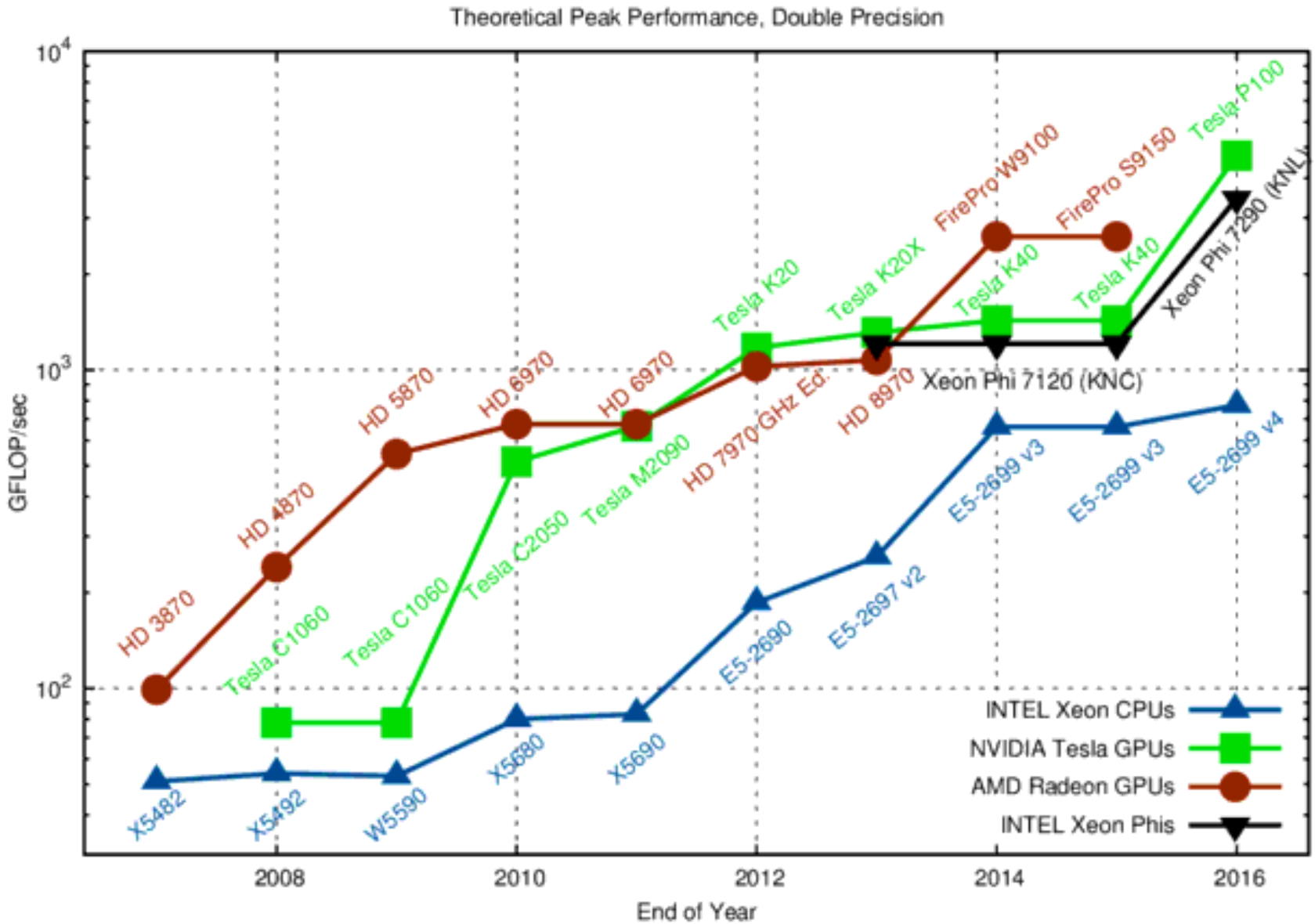
Un peu d'histoire

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

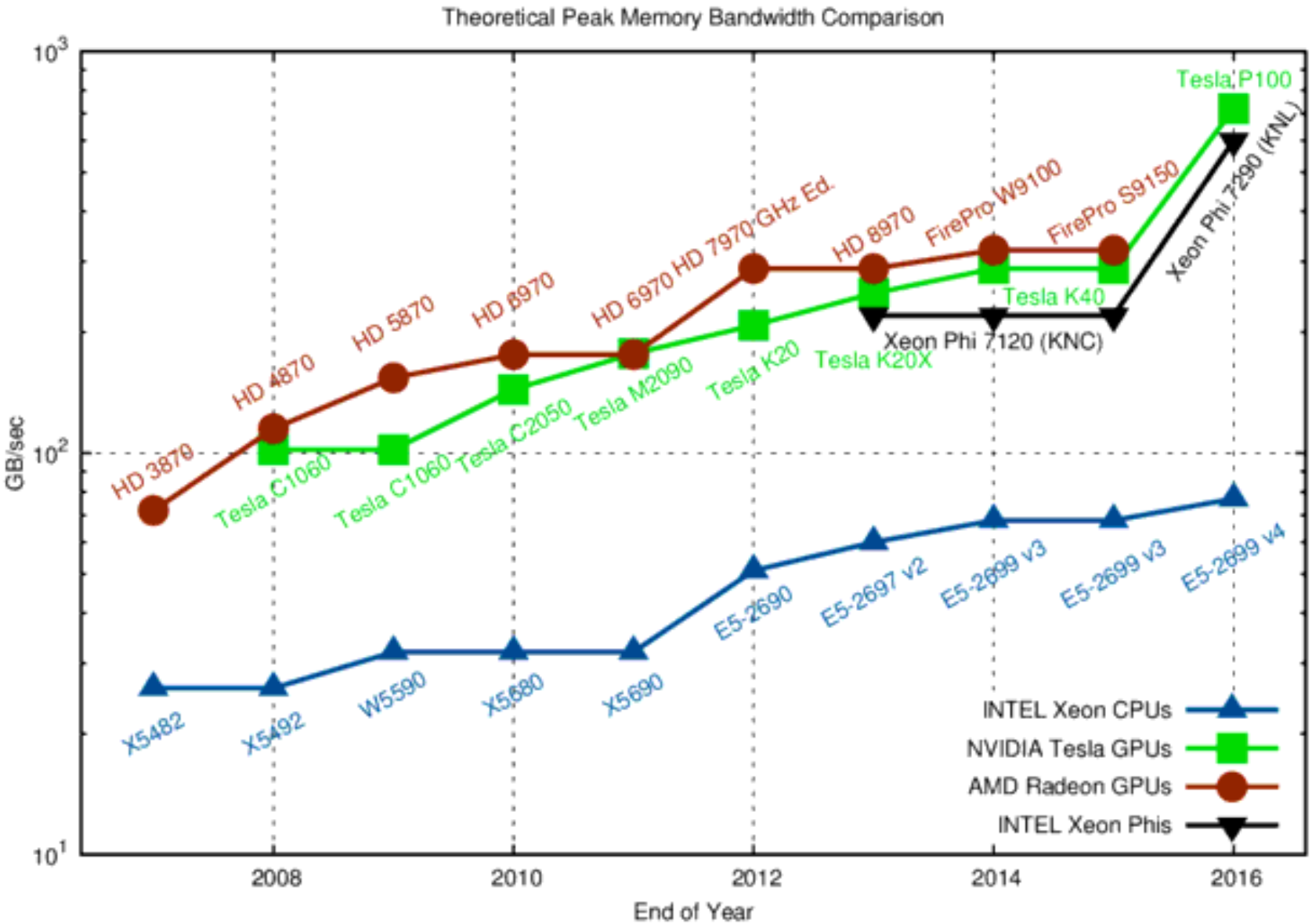


Plan

GPU VS CPU


Theoretical Peak Performance, Double Precision
GFLOP/sec
End of Year
2008
2010
2012
2014
2016
HD 3870
HD 4870
HD 5870
HD 6970
HD 6970
HD 7970 GHz Ed.
HD 8970
FirePro W9100
FirePro S9150
Tesla K20
Tesla K20X
Tesla K40
Tesla K40
Tesla P100
Xeon Phi 7290 (KNL)
Xeon Phi 7120 (KNC)
Tesla C1060
Tesla C1060
Tesla C2050
Tesla M2090
X5482
X5492
W5590
X5680
X5690
E5-2690
E5-2697 v2
E5-2699 v3
E5-2699 v3
E5-2699 v4
INTEL Xeon CPUs
NVIDIA Tesla GPUs
AMD Radeon GPUs
INTEL Xeon Phis

GPU VS CPU


Theoretical Peak Memory Bandwidth Comparison
GB/sec
End of Year
2008
2010
2012
2014
2016
HD 3870
HD 4870
HD 5870
HD 6970
HD 7970 GHz Ed.
HD 8970
FirePro W9100
FirePro S9150
Tesla C1060
Tesla C1060
Tesla C2050
Tesla M2090
Tesla K20
Tesla K20X
Tesla K40
Tesla P100
Xeon Phi 7120 (KNC)
Xeon Phi 7290 (KNL)
X5482
X5492
W5590
X5680
X5690
E5-2690
E5-2697 v2
E5-2699 v3
E5-2699 v3
E5-2699 v4
INTEL Xeon CPUs
NVIDIA Tesla GPUs
AMD Radeon GPUs
INTEL Xeon Phis

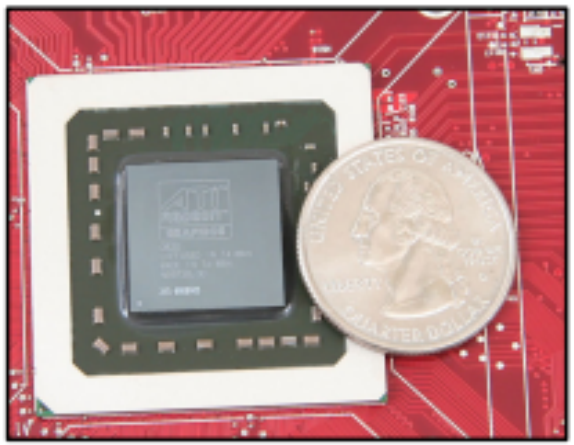
GPU VS CPU

Intel i7 Quad-Core

$\sim$ 100 GFLOPS Peak

730 millions de transistors

4 threads + SSE vector instructions



AMD Radeon HD 587

$\sim$ 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

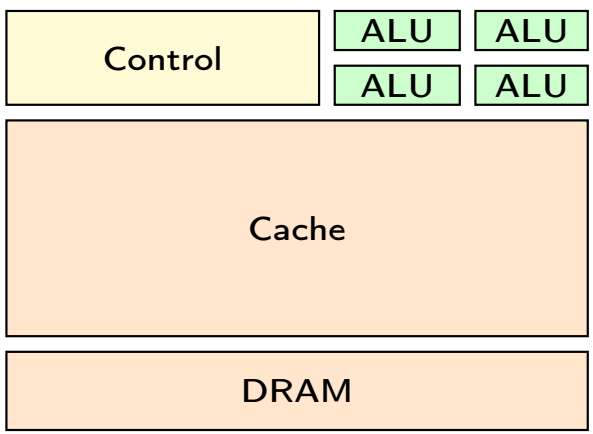
- Les CPU doivent minimiser la latence (négligeant le throughput ☹)
  - Un input du clavier
- Nécessité d'utiliser des caches
- Les CPUs maximisent les opérations en dehors du cache (pre-fetch, exécution out-of-order, ...)

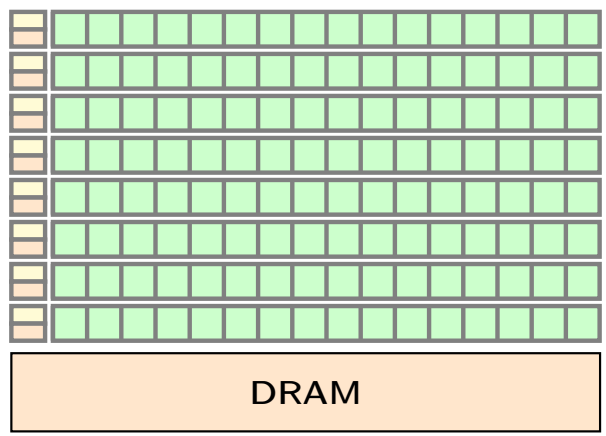
→ Les CPU ont besoin de caches de grande taille

- 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

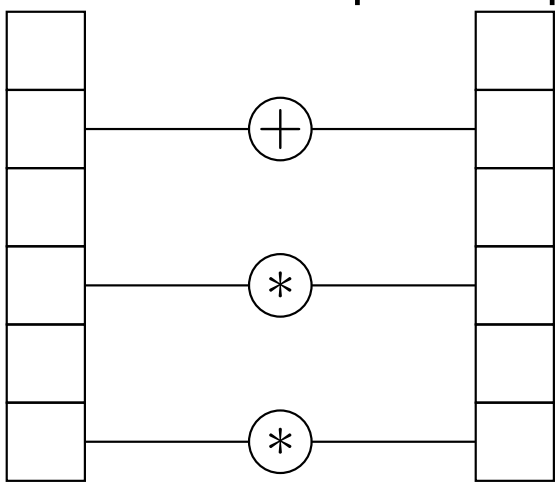
Gestion des threads sur le GPU

- **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

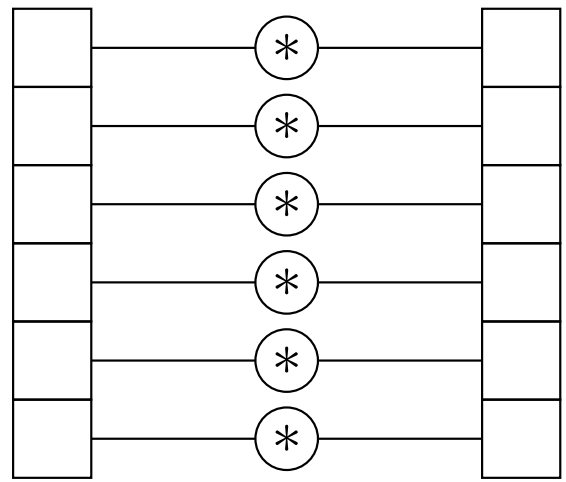
CPU : Parallélisme de tâches

- 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 opération (*"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

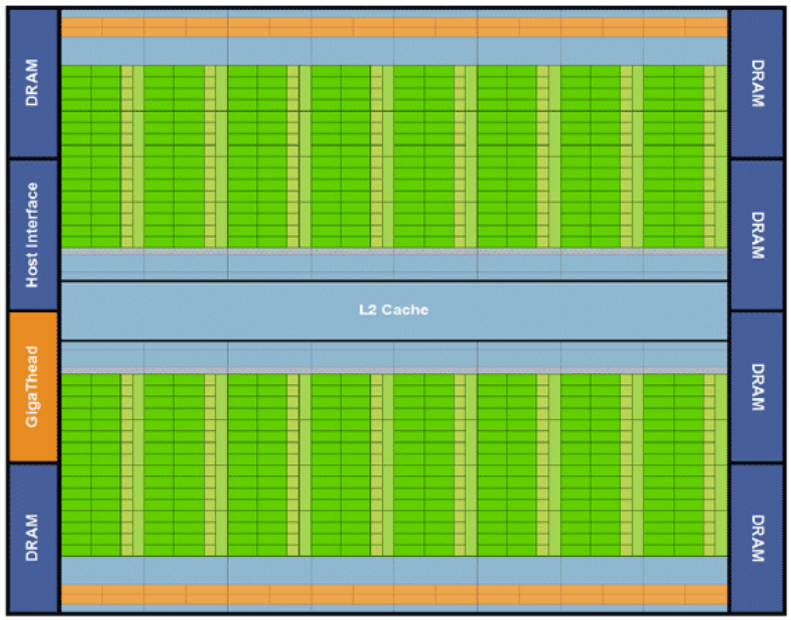
→ Plus de place pour les unités de calcul

- 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

Vu général du GPU Fermi

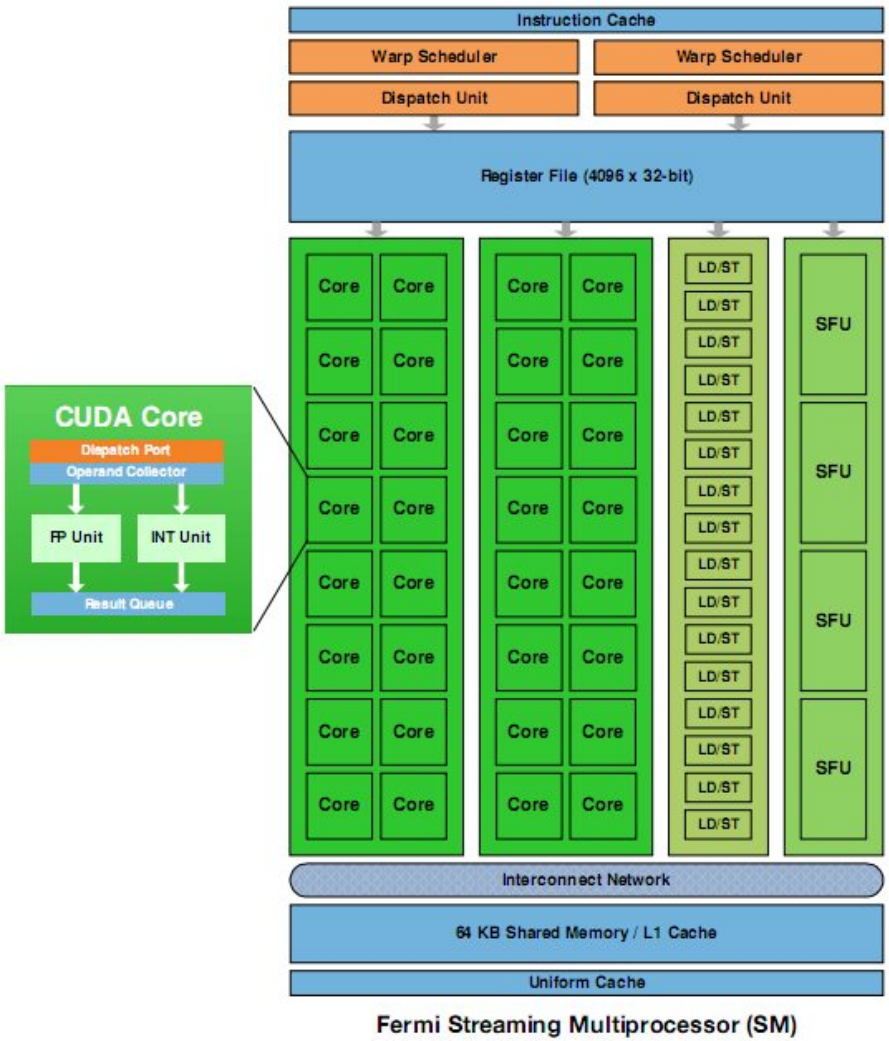
- 512 CUDA cores
  - 1 CUDA core = 1 opération flottante par clock pour un thread
- 16 *Stream Process (SM)* de 32 cœurs
- 6x64-bits mémoire : total 6GB de DRAM
- Connexion avec CPU via PCI express
- GigaThread global scheduler distribue les threads au scheduler de thread sur les SM



Fermi's 16 SM are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).

Vue détaillée du GPU Fermi

- 512 CUDA cores
  - ALU et FPU (*floating point unit*)
- 16 unités Load/Store (LD/ST) par SM
- 4 *Special Function Units* (SFU)
  - sin, cosin, racine carrée, ...
- Les SM ordonnancent les threads par groupe de 32 appelé *warp*
  - Tous les threads d'un warp sont synchronisés
  - Les threads de wrap différents sont indépendants
- 64 Ko mémoire partagée/cache L1



Fermi Streaming Multiprocessor (SM)

Référence : NVIDIA Fermi Compute Architecture Whitepaper

Plan

Programmation GPU

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

- **CUDA**
  - C, C++, Fortran, Python
- OpenCL
- 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
- `https://docs.nvidia.com/cuda/`