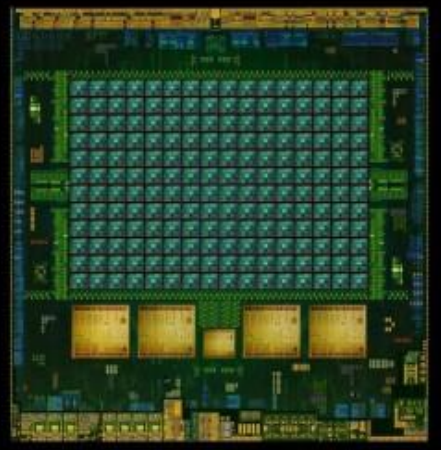
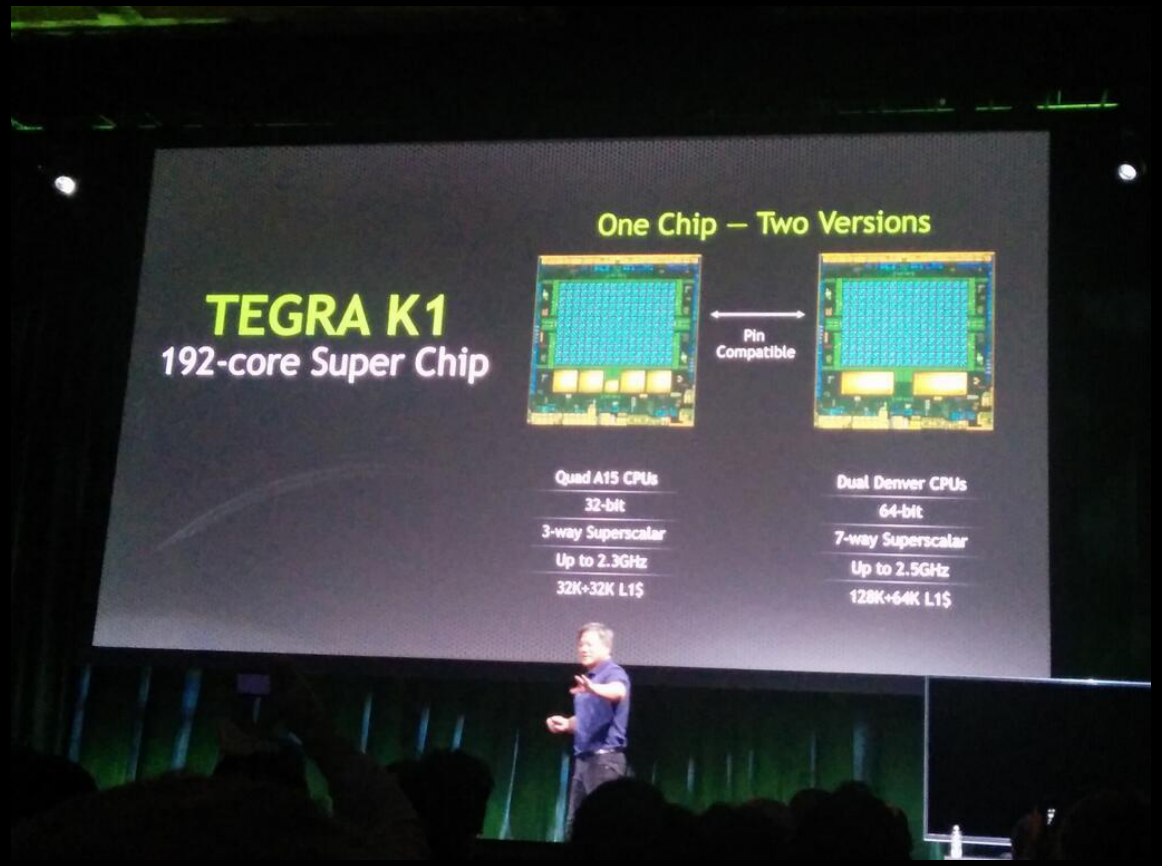
ARCHITECTURE NVIDIA TEGRA K1



Maxime Duchêne
Guillaume Martin
Romain Révol
Joffrey Perrin

Introduction

- Tegra K1 présenté le 6 janvier 2014



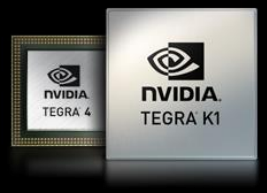
Sommaire

Information générales

- Spécifications

	32 bits	64 bits
Cœurs ARM	4 Quad Core Nvidia+1 ARM Cortex A15	2 Denver Core
Architecture ARM	32 bits Cortex A15	64 bits ARM v8 (Denver)
Architecture GPU	Kepler	Kepler
Cœurs GPU	192 cœurs CUDA	192 cœurs CUDA
Fréquence	2,3GHz	2,5GHz
Cache	32K + 32K L1	128K + 64K L1
Technologie	28nm	
Mémoire	8GB	
Mesure du pic	326 GFLOPS	
Mesure SGEMM	290 GFLOPS	
Performance par Watt	26 GFLOPS/W	

Tegra Road



CPU Architecture	TEGRA 2	TEGRA 3	TEGRA 4	TEGRA K1
Fréquence	Dual Core ULP GeForce	Quad Core ULP GeForce	Quad Core ARM Cortex A15	NVIDIA 4-Plus Quad Core ARM Cortex A15
Cache	Up to 1.2 GHz 32K L1	Up to 1.7 GHz 32K L1	Up to 2 GHz 32K L1	Up to 2.3 GHz 32K L1
technologie	40nm	40nm	28nm	28nm

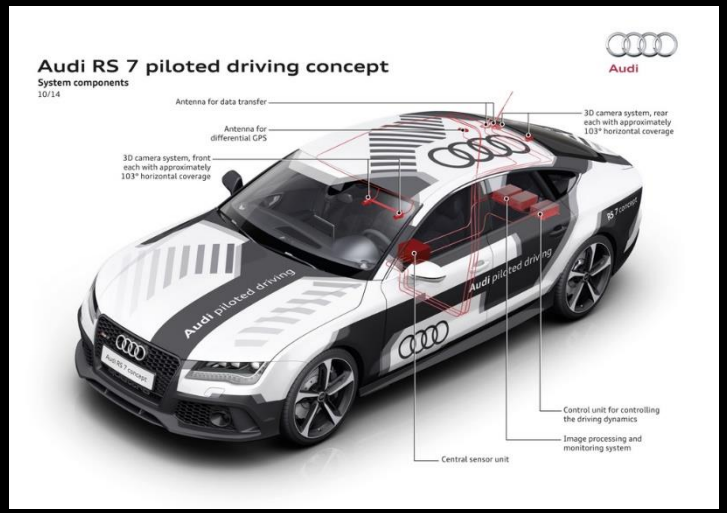
Comparaison

	Exynos 7 Octa (5433)	Snapdragon 805	Tegra K1
Cœur CPU	4x Cortex-A57 + 4x Cortex-A53	4x Krait 450	2x Nvidia Denver
Horloge CPU	4x 1.9GHz + 4x 1,3GHz	4x 2,7GHz	2x 2,5GHz
GPU	MALi-T760	Adreno 420	192 CUDA Core Kepler
Horloge GPU	695MHz	600MHz	950MHz
Mémoire	LPDDR3	LPDDR3	LPDDR3
64 bit?	oui (confirmé)	Non	oui
technologie	20nm	28nm	28nm
Camera max	inconnu	2x 55MP	2x 20MP
Affichage max	1600p	2160p	2160p

Potentiel d’application

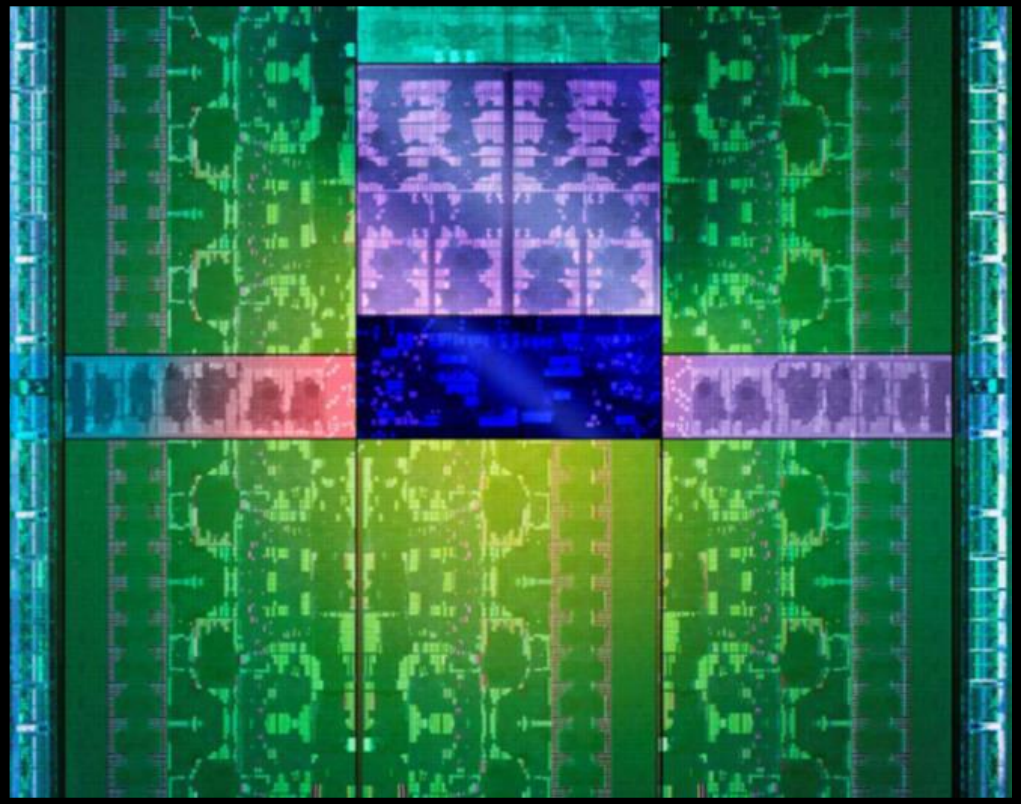
- Processeur mobile pour smartphone et tablettes

- Véhicule autonomes



Architecture : Kepler

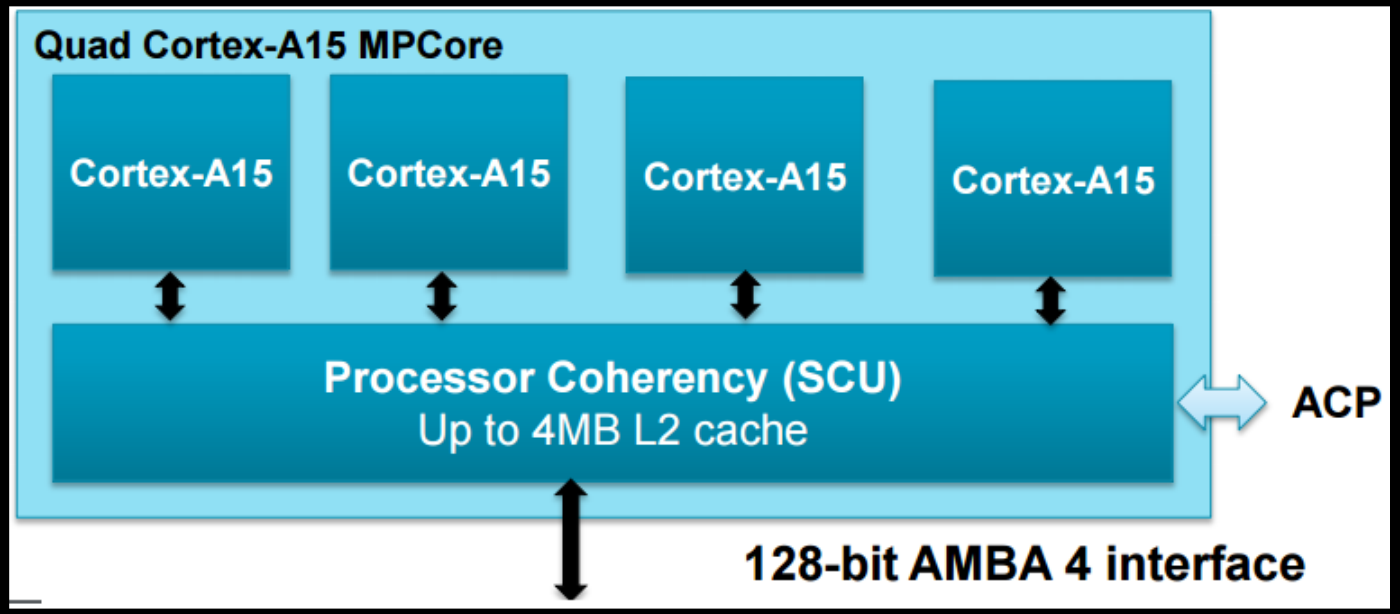
- Conçu pour les performances à faible consommation d'énergie
- 7,1 milliards de transistors
- Débit > 1TFLOP
- Parallélisme dynamique
- Hyper-Q avec GK110 Grid Management Unit (GMU)
- Nvidia GPUDirect RDMA



Architecture : Streaming Multiprocessor(SMX)

- Kepler GK110
  - 15 unités SMX
  - 6 contrôleurs mémoire 64 bits.
- 192 noyaux une précision
- 64 unités double précision
- 32 unités de fonctions spéciales (SFU)
- 32 unités de charge / stockage (LD / ST)

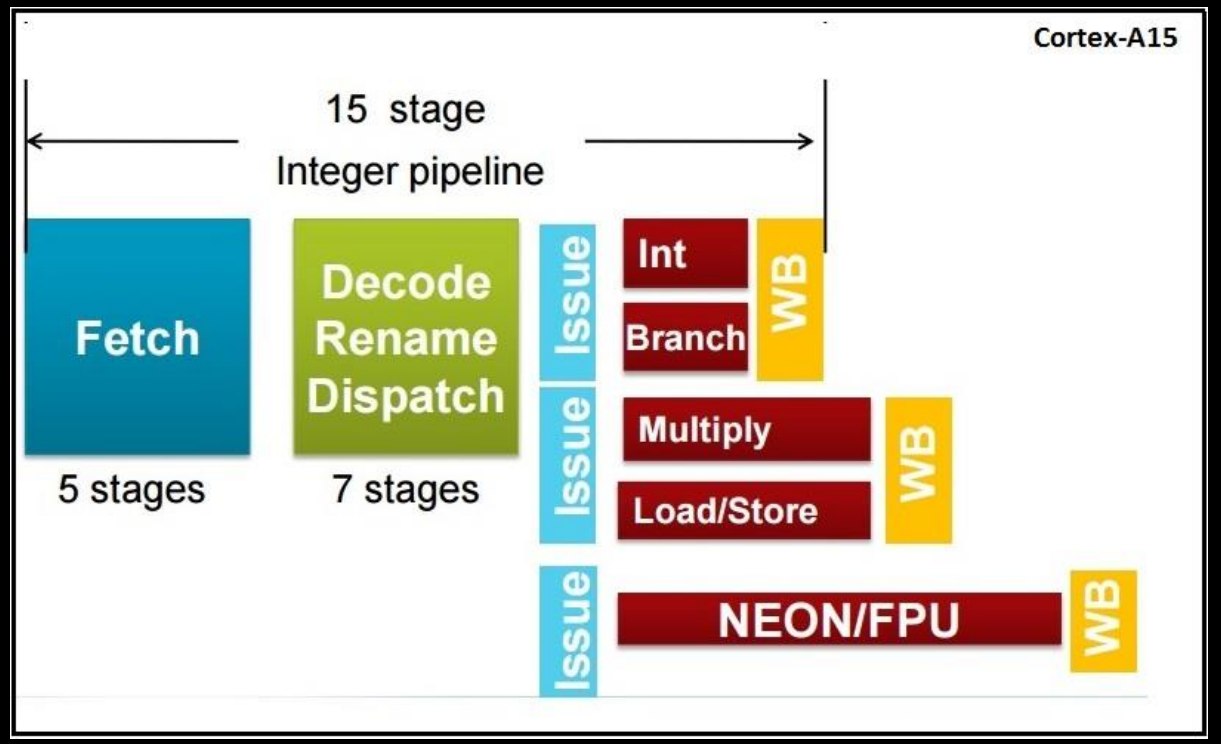
ARM cortex A-15 MPCore



- Cache L1 32Ko dans chaque cœur
- Cache L2 4Mo partagé par tout les cœurs

ARM cortex A-15 MPCore

- Architecture du cœur
  - 1 unité de calcul en virgule flottante
  - 1 DSP
  - 1 accélérateur de calcul NEON
  - 1 unité SIMD



ARM cortex A-15 MPCore

- Optimisation
  - Exécution spéculative :
    - Prédiction du branchement et exécution des instructions
- Jeux d’instructions
  - Thumb-2 utilisé pour réduire la taille des programmes
  - ThumbEE pour la compilation à la volée

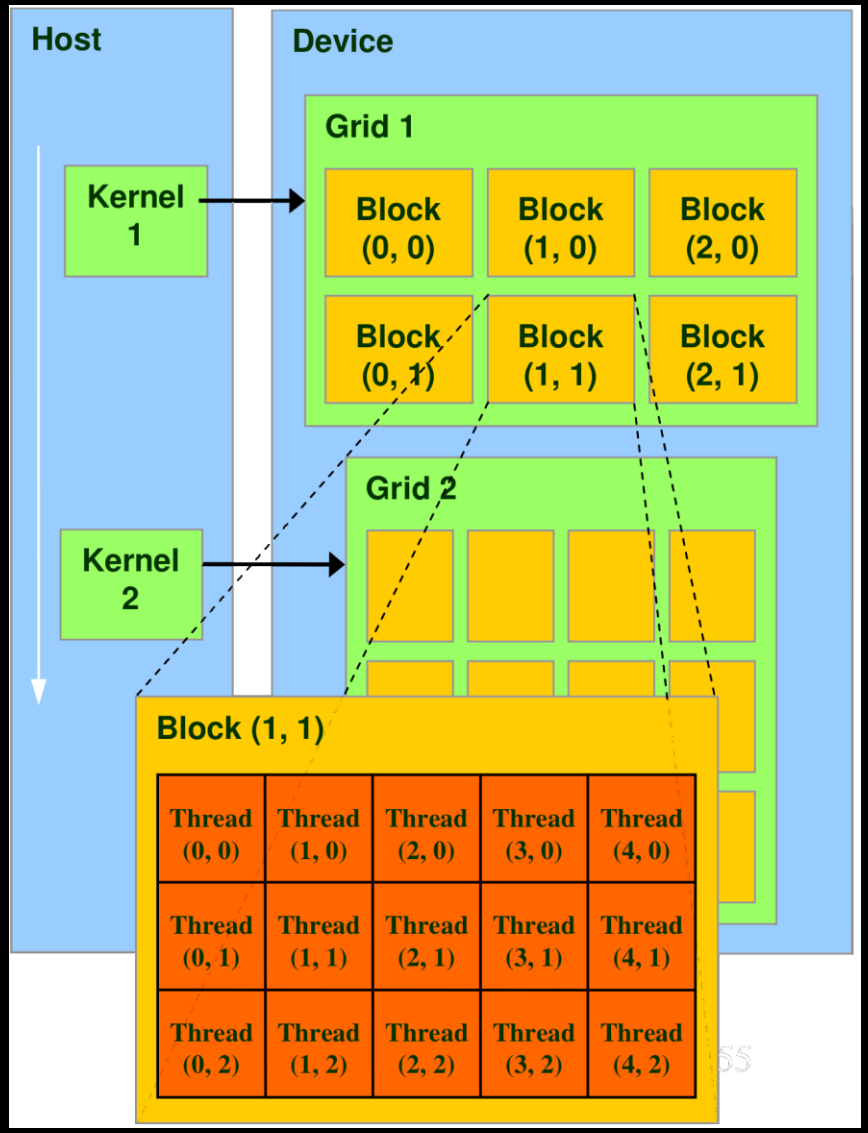
CUDA (Compute Unified Device Architecture)

- Librairie C/C++ et Fortran
- Préfix de fonctions:
  - __host__ : fonction exécuté par le CPU
  - __global__ : fonction exécuté par le GPU mais appelé par le CPU (kernel)
  - __device__ : sous fonction exécuté par le GPU appelé depuis un kernel

- Compilation (windows):
  - nvcc –compile –compiler-options –O2 –o sum.o sum.cu
  - nvcc –link –compiler-options –O2 –o sum.exe sum.o

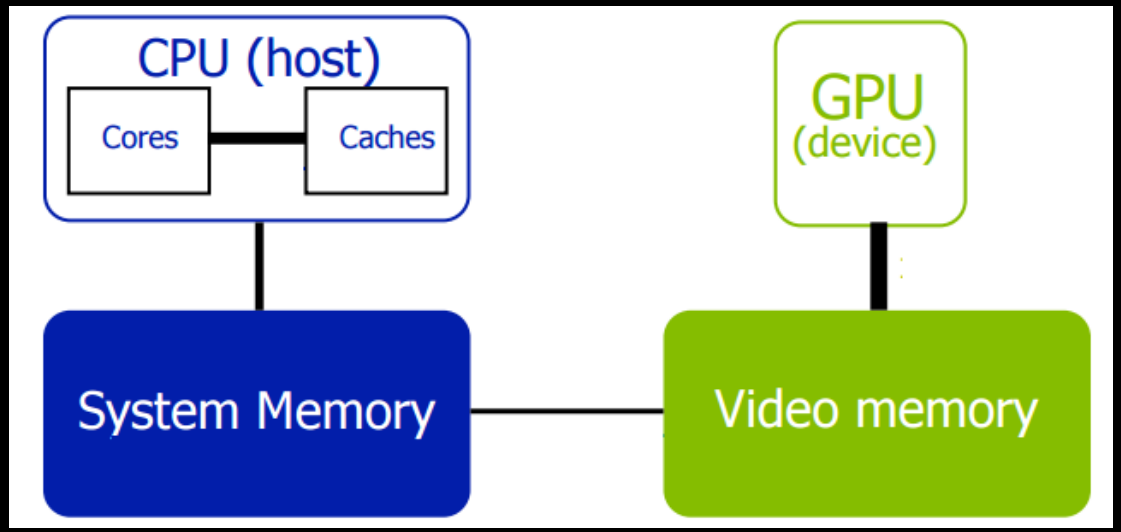
CUDA : threads

- Host appelle une fonction avec un kernel
- Device constitué de grille



CUDA : Etapes

- CPU alloue des données en mémoire (host)
- GPU alloue de la mémoire (device)
- Copie des données en entrée de host vers device
- Execution du Kernel
- Copie des données en sortie de device vers host
- GPU libère la mémoire
- CPU libère la mémoire



Exemple : somme de vecteur

Code C

```c
1. // compute z[i] = x[i] + y[i] for i in [0..size-1]
2. void sum(float *x, float *y, float *z, int size) {
3.   for (int i = 0; i < size; ++i) {
4.     z[i] = x[i] + y[i]; // paralell part
5.   }
6. }
```

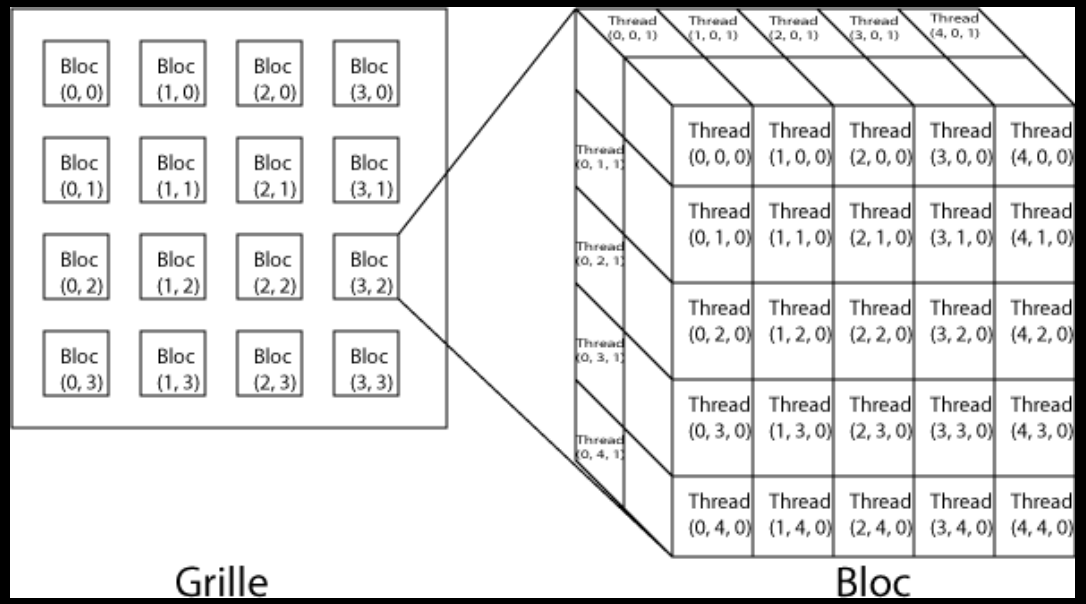
Code CUDA

```c
// ================================
// kernel declaration
// ================================
__global__
void sum(float *x, float *y, float *z, int size) {
  // compute Global Thread InDex
  int gtid = threadIdx.x ;

  if (gtid < size) {
    z[gtid] = x[gtid] + y[gtid]; // paralell part
  }
}
```

Exemple : somme de vecteur

- On souhaite utiliser 1024 threads pour le calcul :
  - On déclare une grille d'un bloc de 1024 threads
    - dim3 grid(1,1,1); dim3 block(1024,1,1);
  - OU une grille d'un bloc de 2x512 threads
    - dim3 grid(1,1,1); dim3 block(512,2,1);
  - OU une grille de 2 blocs de 512 threads
    - dim3 grid(2,1,1); dim3 block(512,1,1);

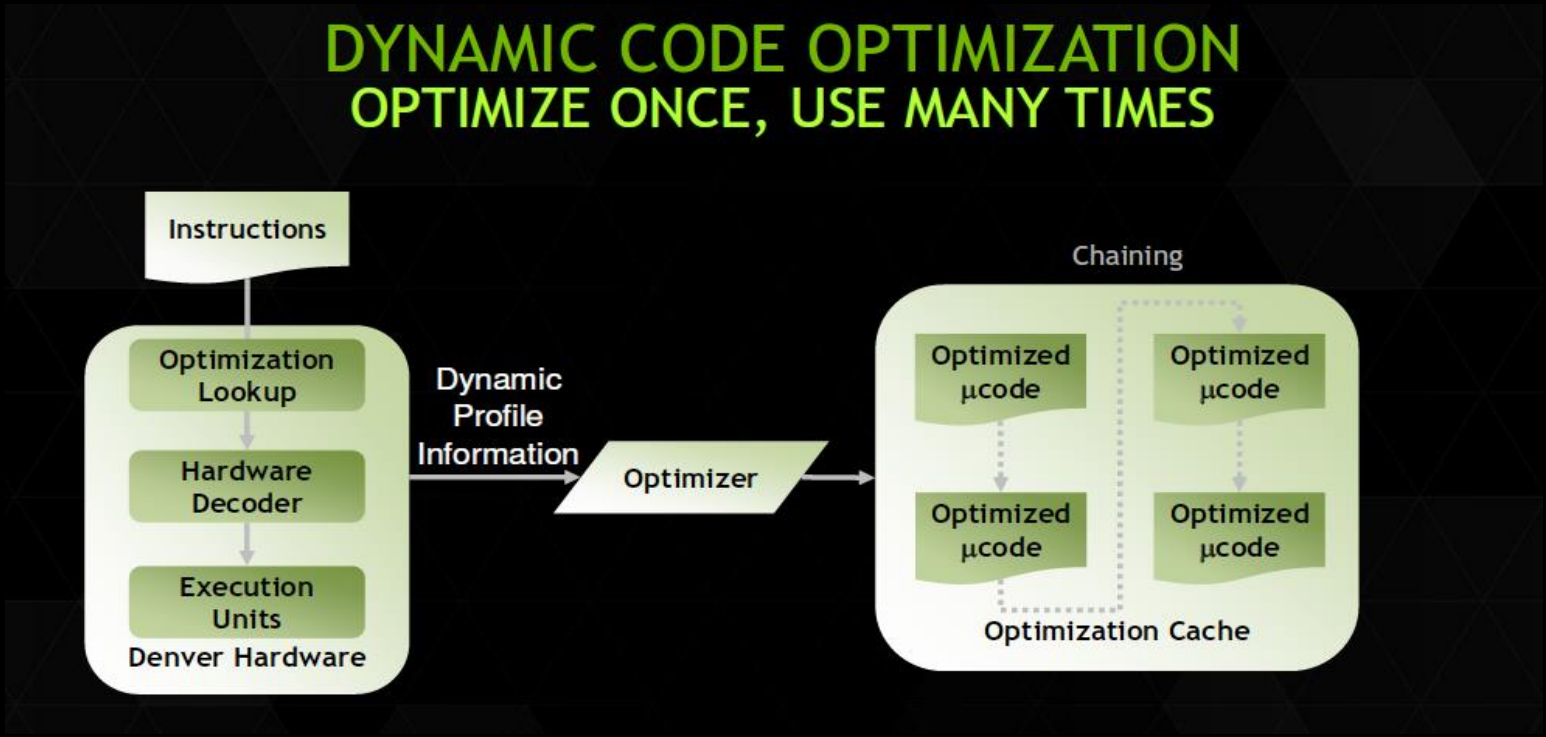


Exemple : somme de vecteur

- Étapes effectuées :
  - Allocation dans la mémoire GPU
  - Copie des données en entrée de host vers device : vecteur x et y
  - Exécution du Kernel
  - Copie des données en sortie de device vers host : vecteur z
  - GPU libère la mémoire

Optimiseur de code dynamique intégré

- Micro-Routine Equivalentes
- Cache d'optimisation de 128MB



Conclusion

- CPU vs GPU
- Tegra X1

Merci pour votre attention