

Disruptif or not disruptif : le GPU

Des myri-alus dans 4cm²

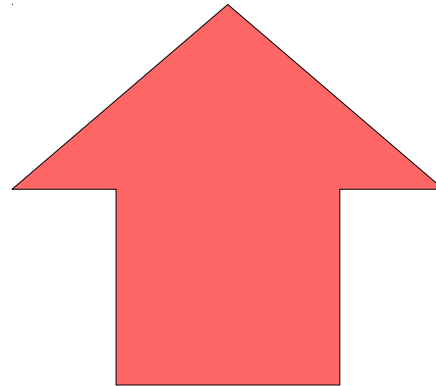
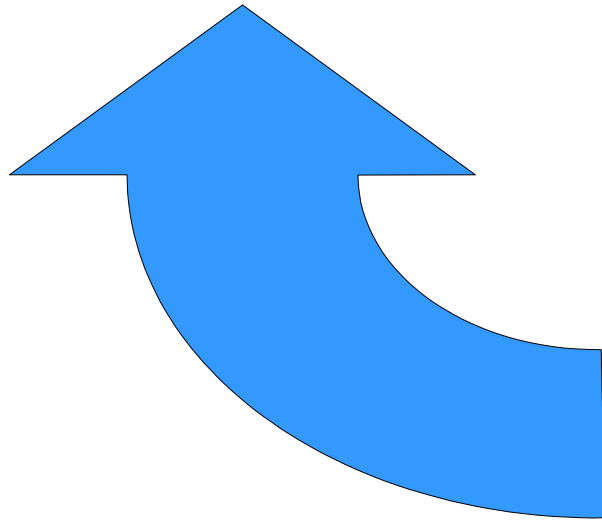
Emmanuel Quémener

Disruptif : “qui sert à rompre”, de disrumpere (rompre)

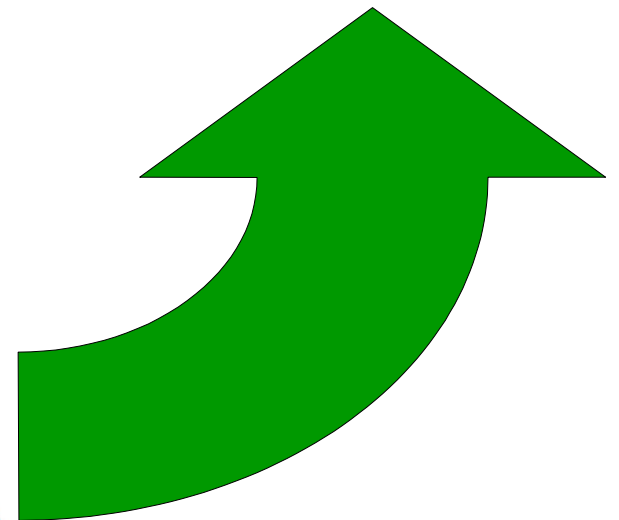
Myriade : “nombre de dix mille”

Blaise Pascal Center « Maison de la modélisation » & ... Conférences

Formations



Projets



Hôtel

CBP

CENTRE BLAISE PASCAL



CBP également centre d'essais

Plate-forme expérimentale avec 11 plateaux techniques

- **Multi-nœuds**: 5 clusters de 4 à 64 nœuds, Nœuds/Cœurs : 4/24, 8/64, 8/64, 64/512
- **Multi-cœurs** : plus de 100 machines de 2 à 28 cœurs,
 - Nœuds/Serveurs : de 8 à 28 cœurs, Workstations : de 2 à 16 cœurs, 16 types de CPU différents
- 
 - 
- **Intégration** : 24 machines virtuelles : Debian de Lenny à Sid sur 32 & 64 bits, ...
- **Exotic hardware** : 3 machines ARMv7 sous Debian Jessie ou Ubuntu
- **Visualisation 3D** : 2 stations , 2 vidéoprojecteurs, 20 moniteurs, 4 lunettes
- **Stations virtuelles 3D** : plus 40 hosts with x2go/VirtualGL
- **COMOD** avec SIDUS : « Compute On My Own Device » pour utilisateurs de labos
 - SIDUS est « Single Instance Distributing Universal System »
- **Prototype Galaxy** pour le traitement intensif de données biologiques

Plateau multi-shaders : (GP)GPU

52 modèles différents...

GPU Gamer : 18

- Nvidia GT 8800GS
- Nvidia GTX 560 Ti
- Nvidia GTX 680
- Nvidia GTX 690
- Nvidia GTX Titan
- Nvidia GTX 780
- Nvidia GTX 780 Ti
- Nvidia GTX 750
- Nvidia GTX 750 Ti
- Nvidia GTX 960
- Nvidia GTX 970
- Nvidia GTX 980
- Nvidia GTX 980 Ti
- Nvidia GTX 1050 Ti
- Nvidia GTX 1060
- Nvidia GTX 1070
- Nvidia GTX 1080
- Nvidia GTX 1080 Ti



GPGPU : 9

- Nvidia Tesla C1060
- Nvidia Tesla M2050
- Nvidia Tesla M2070
- Nvidia Tesla M2090
- Nvidia Tesla K20m
- Nvidia Tesla K40c
- Nvidia Tesla K40m
- Nvidia Tesla K80
- Nvidia Tesla P100

GPU desktop & pro : 13

- Nvidia Quadro 600
- Nvidia Quadro 4000
- Nvidia Quadro K2000
- Nvidia Quadro K4000
- Nvidia Quadro FX4800
- Nvidia NVS 310
- Nvidia NVS 315
- Nvidia GT 430
- Nvidia GT 620
- Nvidia GT 640
- Nvidia GT 710
- Nvidia GT 730
- Nvidia GT 1030



GPU AMD Gamer & al : 12

- AMD Radeon HD 5850
- AMD Radeon HD 7970
- AMD R9 380
- AMD R9 290X
- AMD R9 295x2
- AMD R9 Fury
- AMD Nano Fury
- AMD Radeon HD 6450
- AMD Radeon HD 6670
- AMD R7 240
- AMD Kaveri A10-7850K
- AMD RX Vega 64

Centre Blaise Pascal ~ Dryden FR

Petit exemple illustratif



- Nasa X-29
- Cellule de F-5
- Moteur de F-18
- Train de F-16
- Études de
 - Plans « canard »
 - Incidence $>50^\circ$
 - « Fly-By-Wire »

Recycler, réutiliser & explorer de nouveaux domaines

Quel point de vue ?

Le point de vue d'un physicien...

- Approche « système » du physicien
 - Héritage des ordinateurs analogique
 - Un « système » informatique :
 - réseau / matériel / OS / logiciels / codes / usages / ...
- Approche « Saint Thomas »
 - Appréhension par ma seule mesure
- Pilote d'essai
 - Caractérisation, recherche d'une exploitation optimale

Il y a une génération (humaine)...

Un film de série B de 1984

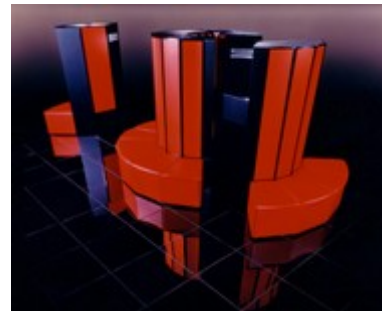
- 1984 : The Last Starfighter

- 27 minutes de CGI
- $22.5 \cdot 10^9$ opérations par image
- 1 Cray X-MP (130 kW)
- 66 jours (en fait, il a fallu 1 an)



- 2017: sur la GTX 1080Ti (250 W)

- 3.4 seconds
- Comparaison GTX1080Ti / Cray
 - Performance : $\times 1600000$!
 - Consommation / $\sim 10^9$

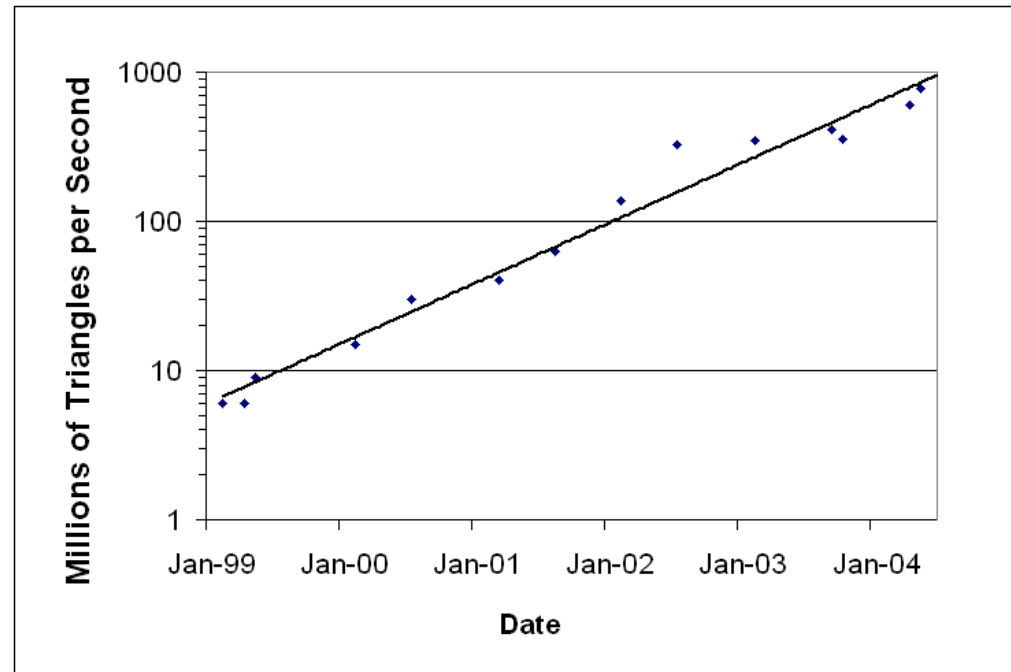


Pourquoi le GPU est « disruptif » ?

Le « grand détournement » en HPC

- Dans une conférence à l'ENS-Cachan 2006Q1, ceci...

- x100 en 5 ans



- Entre 2000 et 2015

- GeForce 2 Go/GTX 980Ti : de 286 à 2816000 MOperations/s : x10000
 - Pour un CPU de station de travail : x100

Au commencement de toutes choses

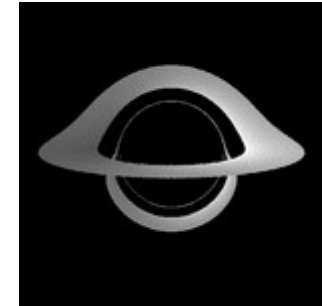
- Nvidia lance le « superordinateur personnel »
- Quand : 2008-11-19 Qui : Nvidia
- Où : sur Tom's Hardware;-)
- Quoi : une carte PCIe Tesla C1060 PCIe avec 240 cœurs
- Combien : 933 Gflops Float32 (mais 78 Gflops Float64)



Pourquoi un GPU est-il si puissant ?

Pour construire une image 3D !

- 2 approches:
 - *Raytracing* : PovRay (« dimensions » de UMPA)
 - *Shading* : 3 opérations
- « Raytracing » :
 - De l'œil au contact de chaque objet
- « Shading »
 - Model2World: objets vectoriels placés dans la scène
 - World2View: objets vectoriels projetés sur un « plan » vectoriel
 - View2Projection: pixellisation du « plan » vectoriel

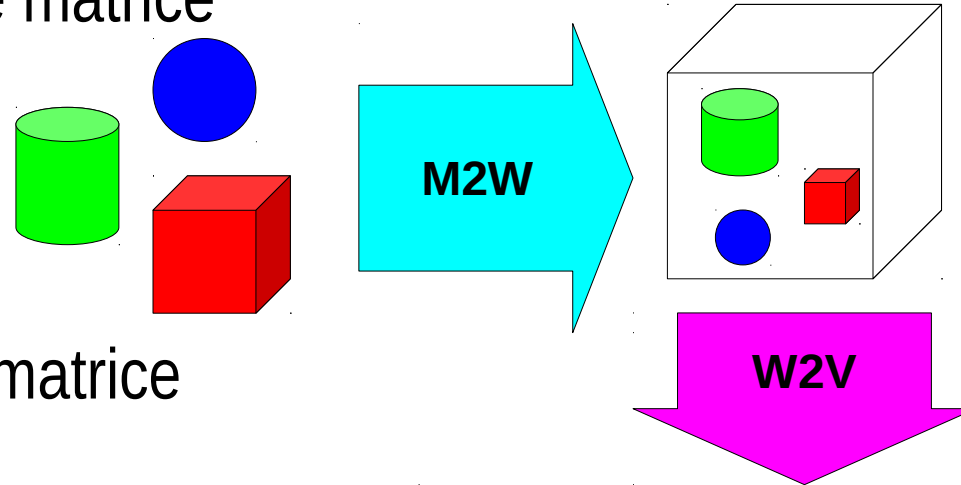


Pourquoi le GPU est-il si puissant ?

Shadering & Calcul Matriciel

- Model 2 World: 3 produits de matrice

- Rotation
- Translation
- Mise à l'échelle

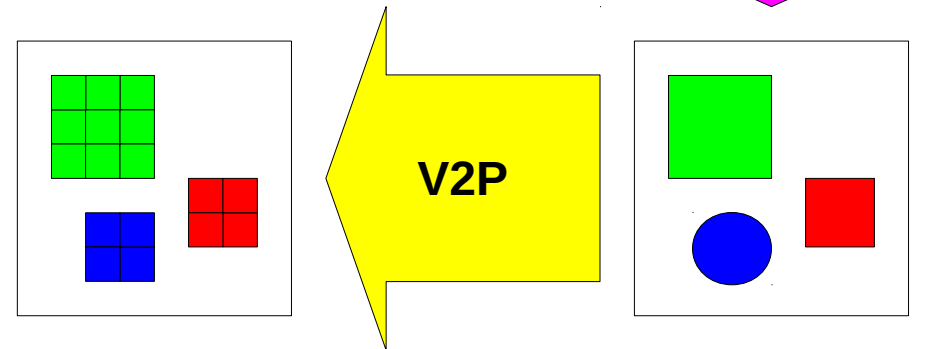


- World 2 View: 2 produits de matrice

- Positionnement de la caméra
- Ligne de vue

- View 2 Projection

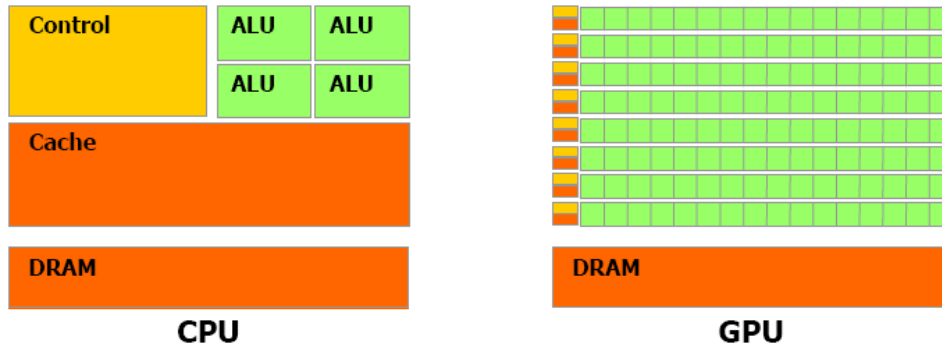
- Pixellisation



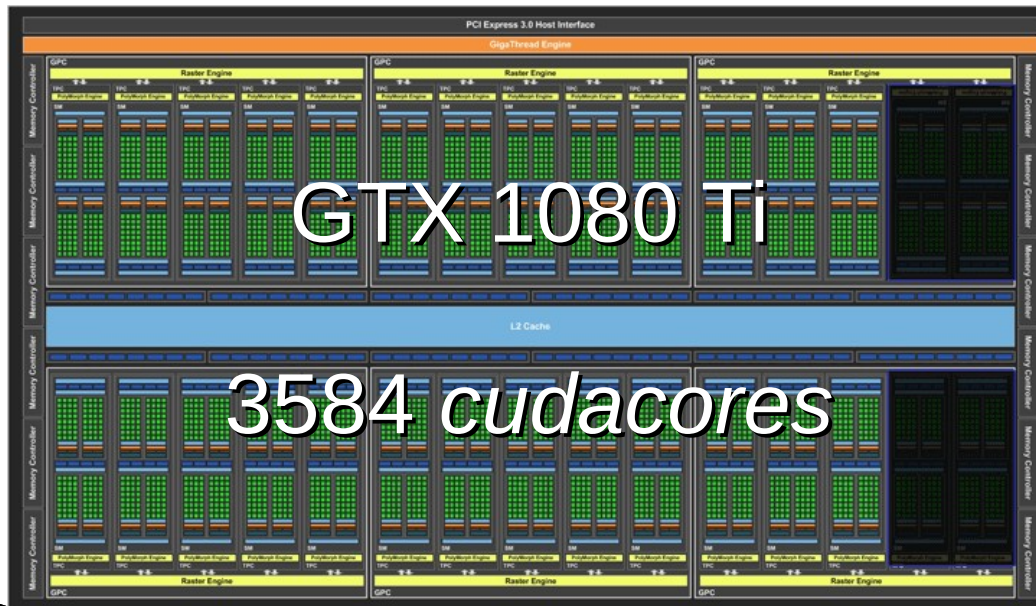
Donc, à l'origine, le GPU, un gros multiplicateur de Matrice

Combien de composants ?

Grosse différence entre GPU/CPU

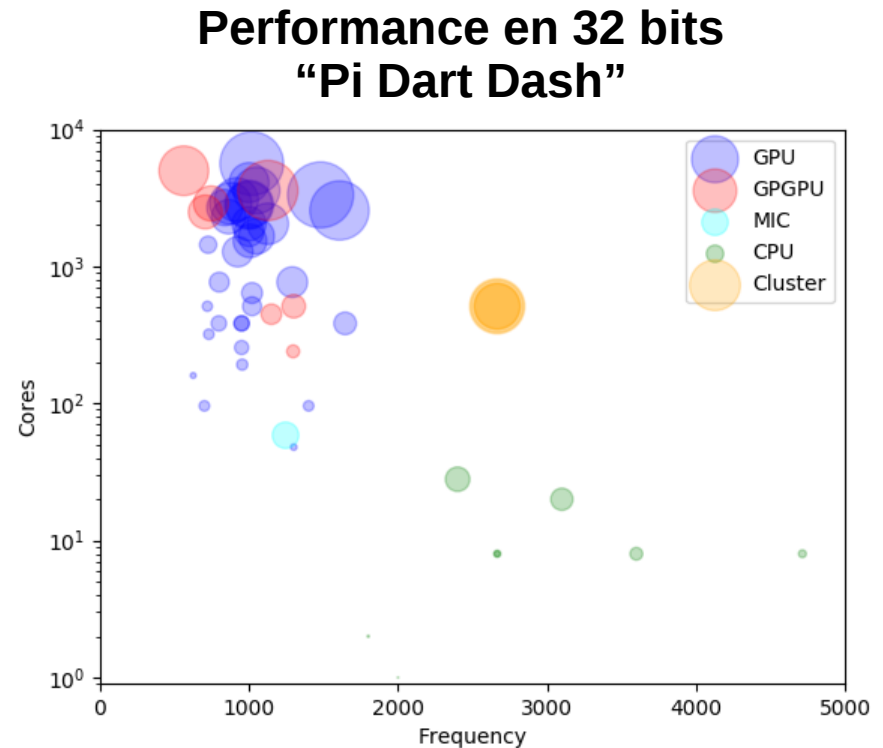
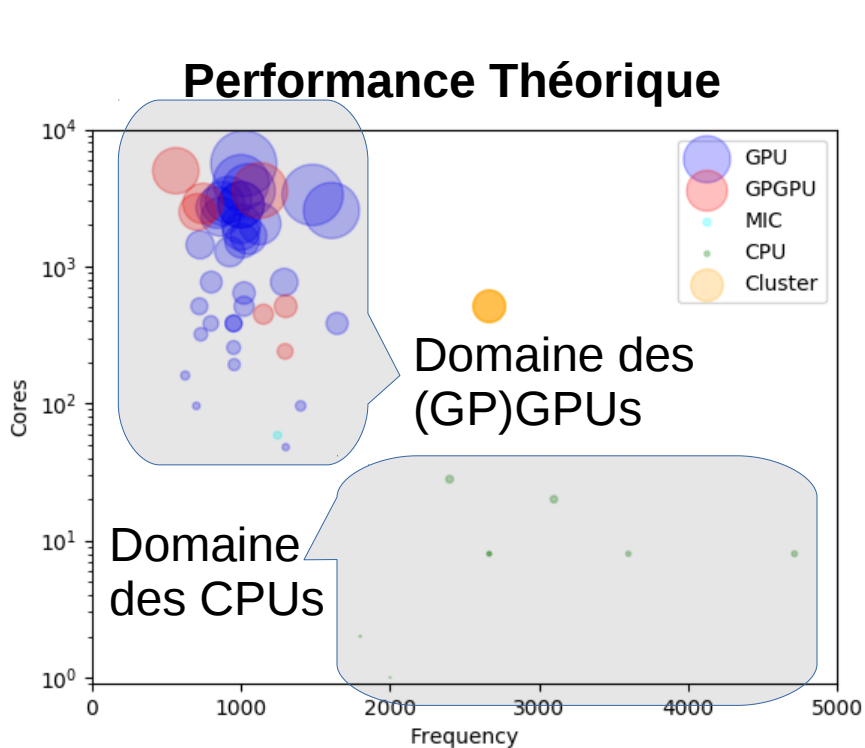


- Opérations
 - Multiplication de matrice
 - vectorisation
 - "Pipelining"
 - Shader (multi) processor
- Implementation : 1993
 - OpenGL, Glide, Direct3D,
- Généricité : 2002
 - CgToolkit, CUDA, OpenCL



Comment différencier CPU/GPU

Question de battement, de cœurs ?



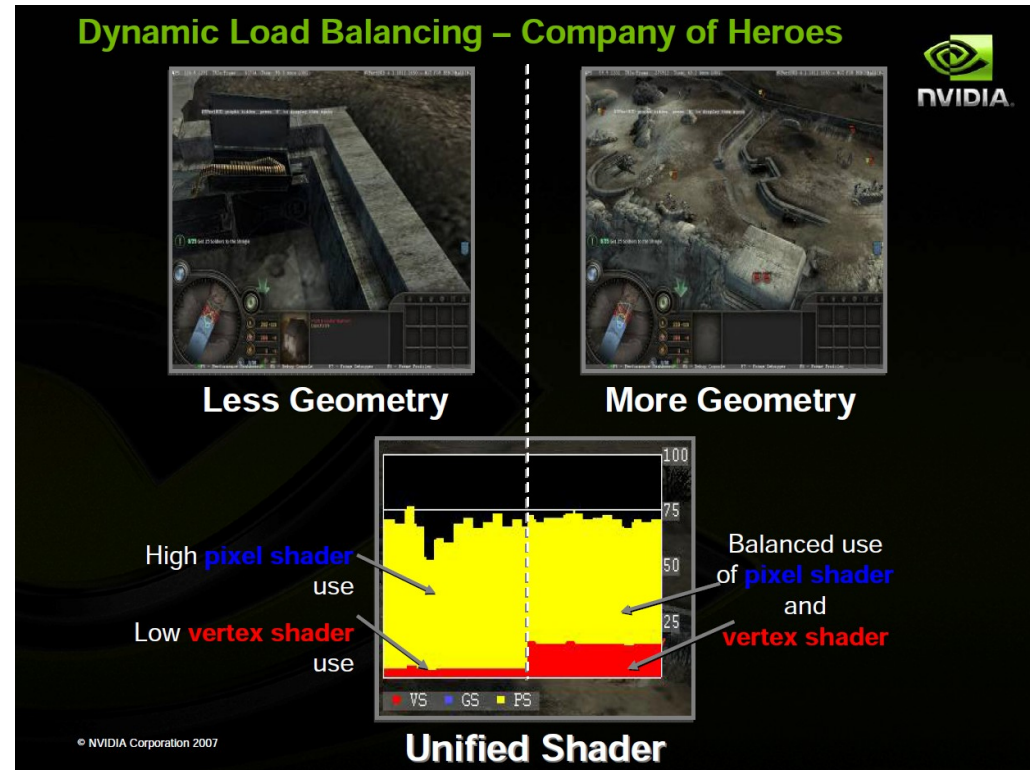
- Sur un GPU, cohérence entre performances théorique & pratiques
- Sur un CPU, performance relative meilleure

Pourquoi un GPU est-il si puissant ?

Le retour à une base générique

Dans le GPU

- Unités très spécialisées
- Efficacité du pipelining
- Perte de généricité
- Scènes variables
- Nature des détails

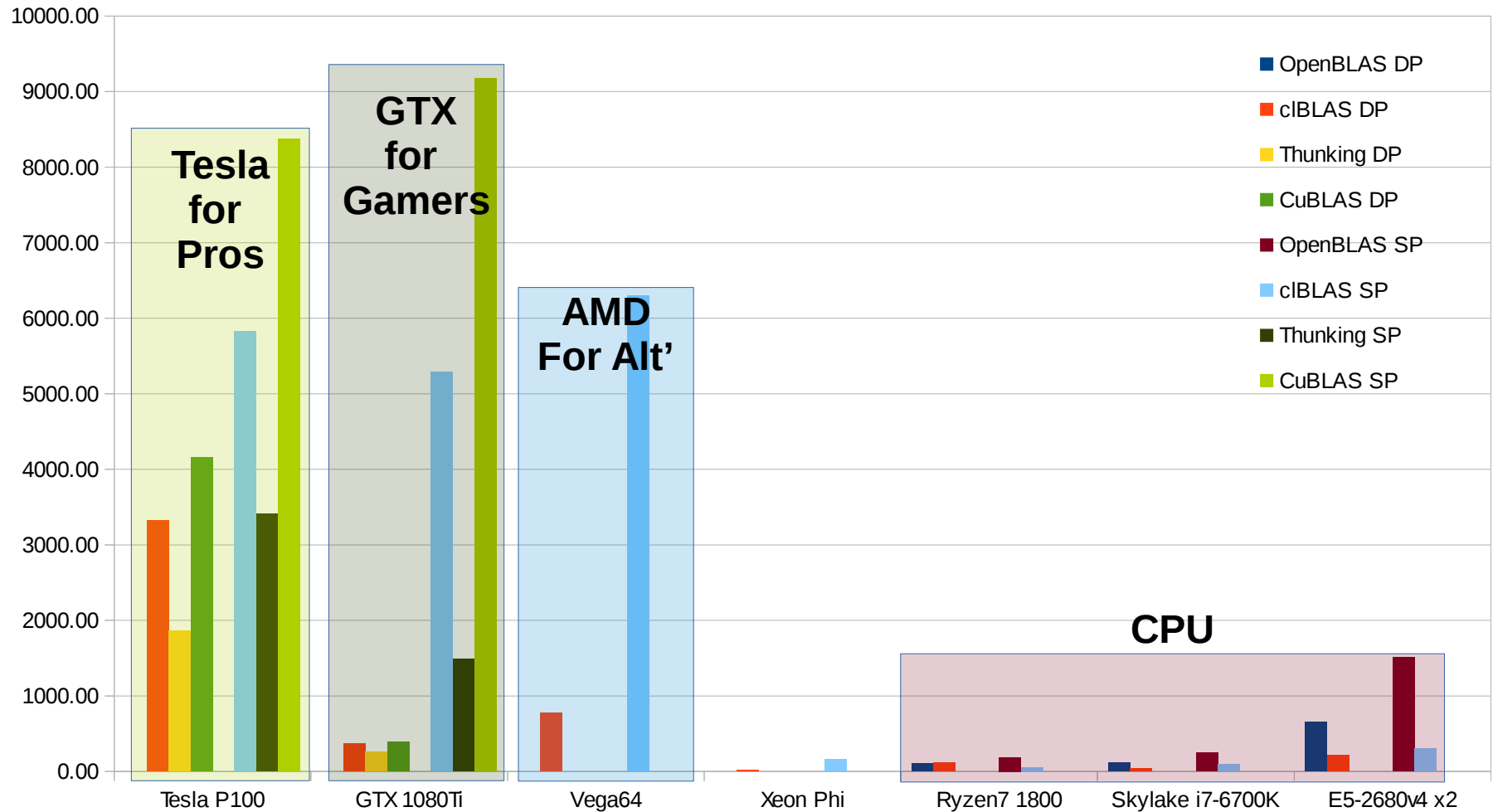


La solution :

Revenir à des fonctions plus générales !

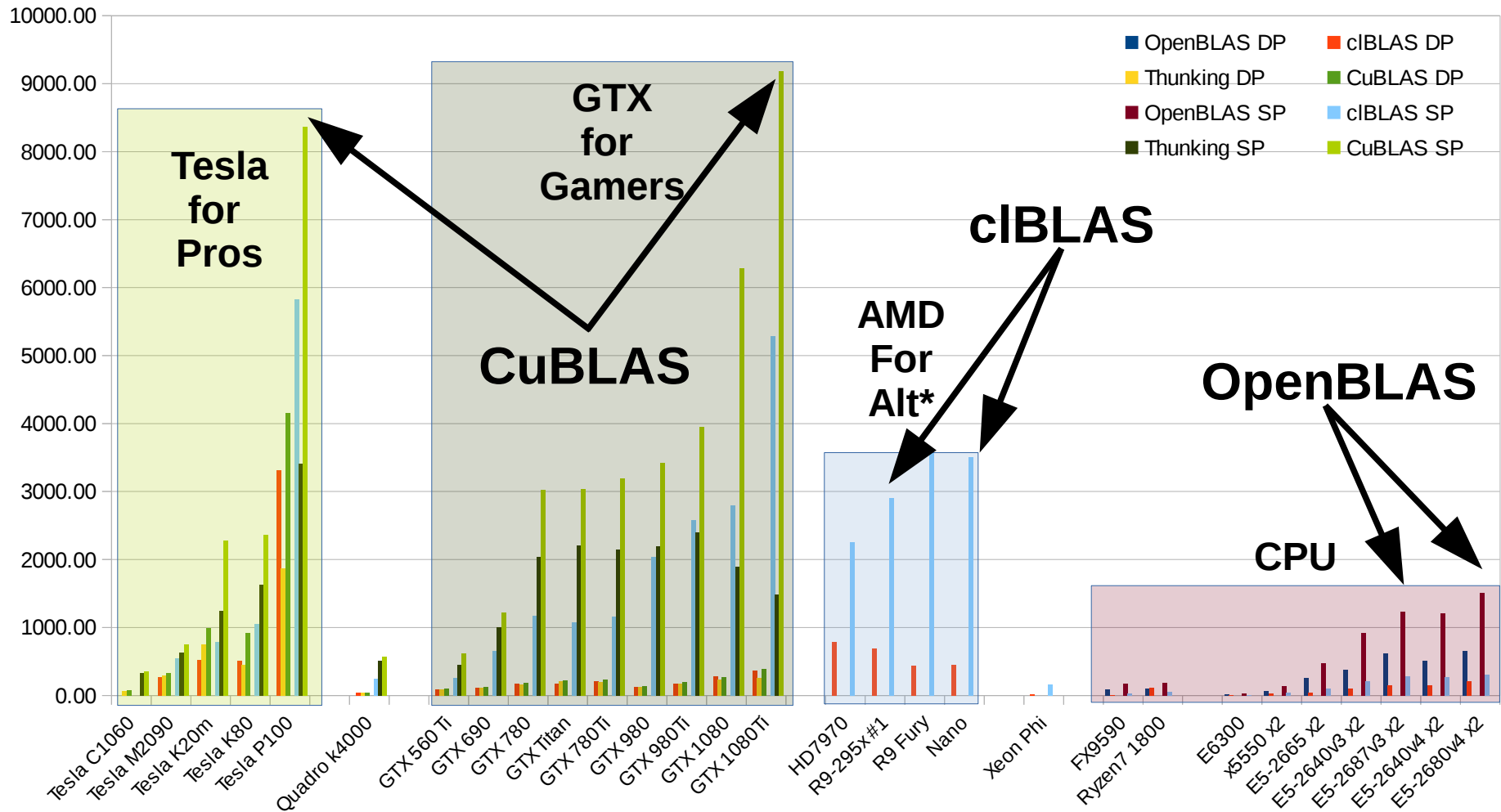
Est-ce que le GPU est si puissant ?

xGEMM pour mes meilleures MyriALUs...



BLAS, le match : CPU vs GPU

xGEMM quand $x=D(P)$ or $S(P)$



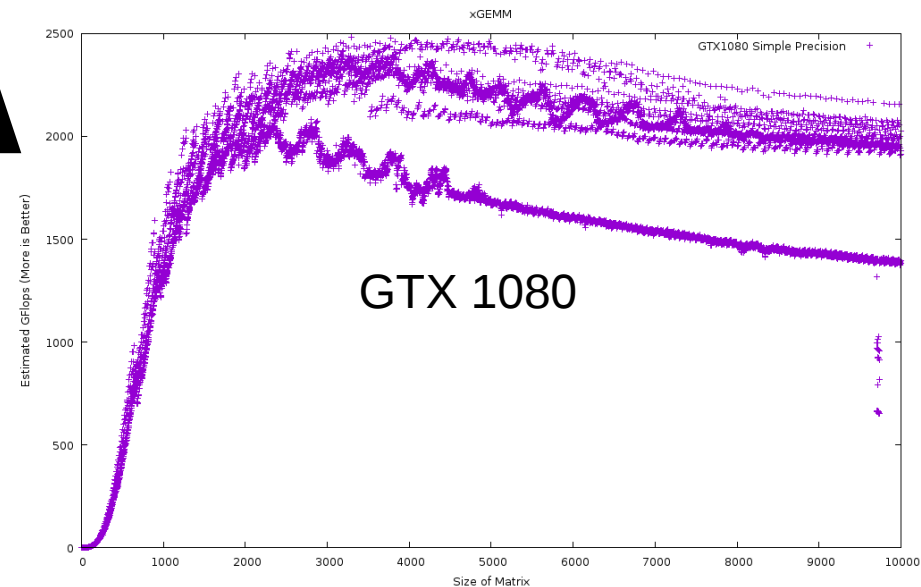
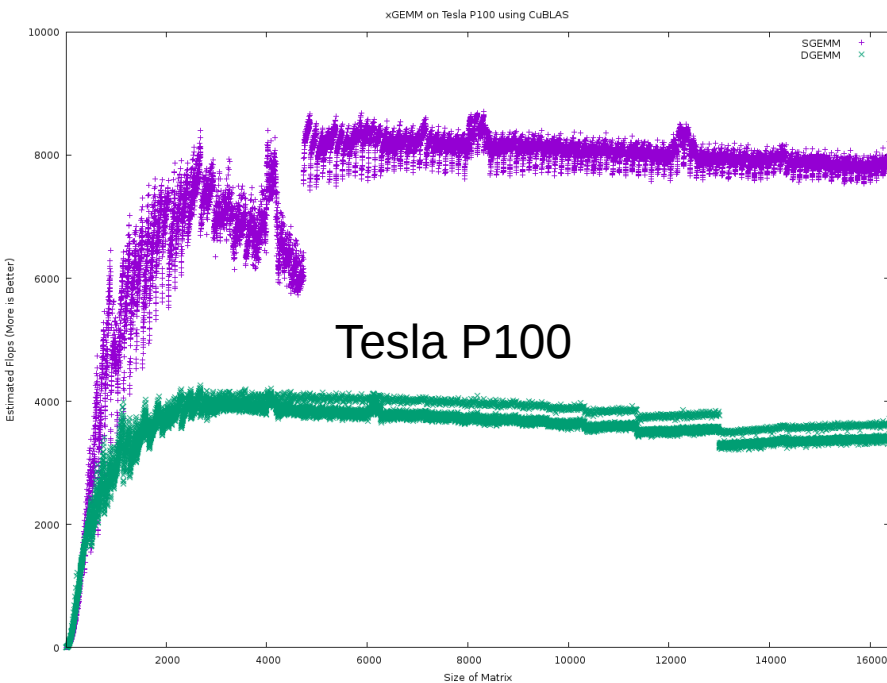
Déjà, en 2010, des trucs étranges ...

La Tesla C1060 ...

- Un produit matriciel : xGEMM
 - Propriété: Transposé ($A * B$) = - Transposé (A) * Transposé (B)
- Résultats (en Gflops): Yess!
 - SP: FBLAS / CBLAS: 12, CuBLAS: 350/327: x27 !!!
 - DP: FBLAS / CBLAS: 6, CuBLAS: 73/70: x11!
- Surprise : Mmmm! CuBLAS préfère the x16!
 - SP: 16000^2 , 350, but 15999^2 or 16001^2 , 97: x3.6!
 - DP: 10000^2 , 73, but 9999^2 or 10001^2 , 31: x2.35

Ce que Nvidia ne diffuse pas... xGEMM sur un large domaine...

Performance



Taille

La performance est là, mais pas tout le temps...

Petit tour d'horizon du parallélisme

Modèles de programmation parallèle

	Cluster	Node CPU	Node GPU	Node Nvidia	Accelerator	FPGA
MPI	Yes	Yes	Non	No	Yes*	No
PVM	Yes	Yes	No	No	Yes*	No
OpenMP	No	Yes	No	No	Yes*	No
Pthreads	No	Yes	No	No	Yes*	No
OpenCL	No	Yes	Yes	Yes	Yes	Yes
CUDA	No	No	No	Yes	No	No
TBB	No	Yes	No	No	Yes*	No

- OpenCL «apparaît» comme le plus polyvalent :-)
- CUDA ne peut être utilisé qu'avec les GPU Nvidia
- Les accélérateurs « semblent » plus agnostiques...

Différences entre CUDA/OpenCL

Le « kernel », le noyau de calcul

```
__device__ ulong MainLoop(ulong iterations,uint seed_w,uint seed_z,size_t work)
{
    uint jcong=seed_z+work;
    ulong total=0;
    for (ulong i=0;i<iterations;i++) {
        float x=CONGfp ;
        float y=CONGfp ;
        ulong inside=((x*x+y*y) <= THEONE) ? 1:0;
        total+=inside;
    }
    __global__ void MainLoopBlocks(ulong *s,ulong iterations,uint seed_w,uint seed_z)
    {
        ulong total=MainLoop(iterations,seed_z,seed_w,blockIdx.x);
        s[blockIdx.x]=total;
        __syncthreads();
    }
```

```
ulong MainLoop(ulong iterations,uint seed_z,uint seed_w,size_t work)
{
    uint jcong=seed_z+work;
    ulong total=0;
    for (ulong i=0;i<iterations;i++) {
        float x=CONGfp ;
        float y=CONGfp ;
        ulong inside=((x*x+y*y) <= THEONE) ? 1:0;
        total+=inside;
    }
    return(total);
}
__kernel void MainLoopGlobal(__global ulong *s,ulong iterations,uint seed_w,uint seed_z)
{
    ulong total=MainLoop(iterations,seed_z,seed_w,get_global_id(0));
    barrier(CLK_GLOBAL_MEM_FENCE);
    s[get_global_id(0)]=total;
}
```

Agir en « intégrateur »

Pour ne pas réinventer la roue...

Librairies de programmation parallèles

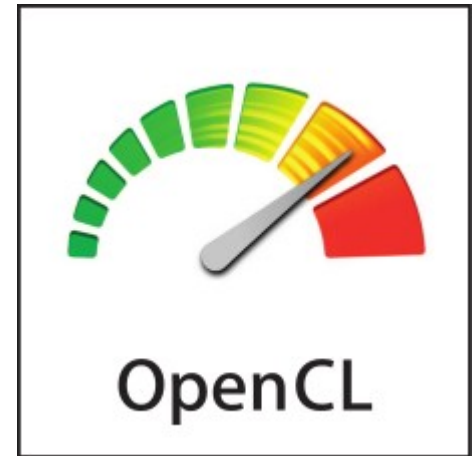
	Cluster	Nœud CPU	Nœud GPU	Nœud Nvidia	Accélérateur
BLAS	BLACS MKL	OpenBLAS MKL	clBLAS	CuBLAS clBLAS	MKL clBLAS
LAPACK	Scalapack MKL	Atlas MKL	clMAGMA	MAGMA	MagmaMIC
FFT	FFTW3	FFTW3	clFFT	CuFFT clFFT	FFTW3 clFFT

- Les librairies classiques utilisables avec les GPU
 - Nvidia fournit plein d'implémentations (BLAS, FFT, ...)
 - OpenCL en fournit certaines, mais pas complètes !

Pourquoi OpenCL ?

Une histoire pas vraiment « politiquement correcte »

- Autour de 2005, MacOSX a besoin de plus de puissance
 - Des traitements peuvent être déchargés au GPU
 - CUDA émerge comme le successeur de CgToolkit
- Mais Apple ne veut pas être prisonnier...
 - (comme ses utilisateurs ;-))
- Apple lance la création du consortium Khronos
 - AMD, IBM et ARM viennent rapidement
 - ... et Intel & Nvidia traînent les pieds mais arrivent...



Ce que OpenCL offre...

10 implémentations sur x86

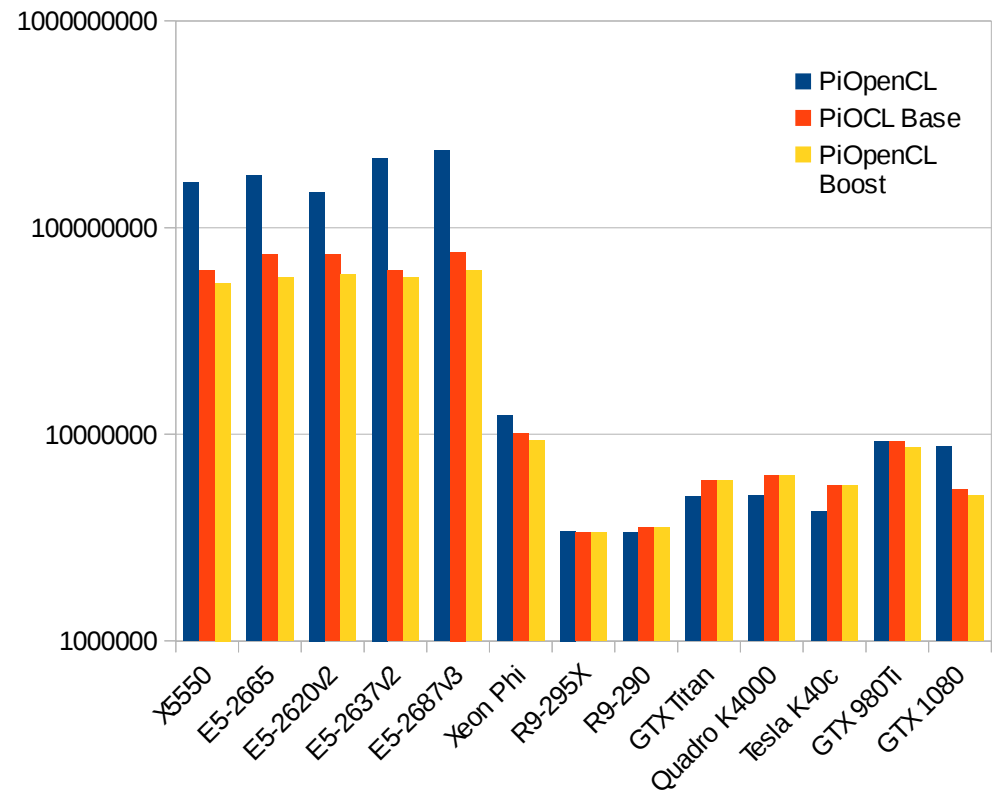
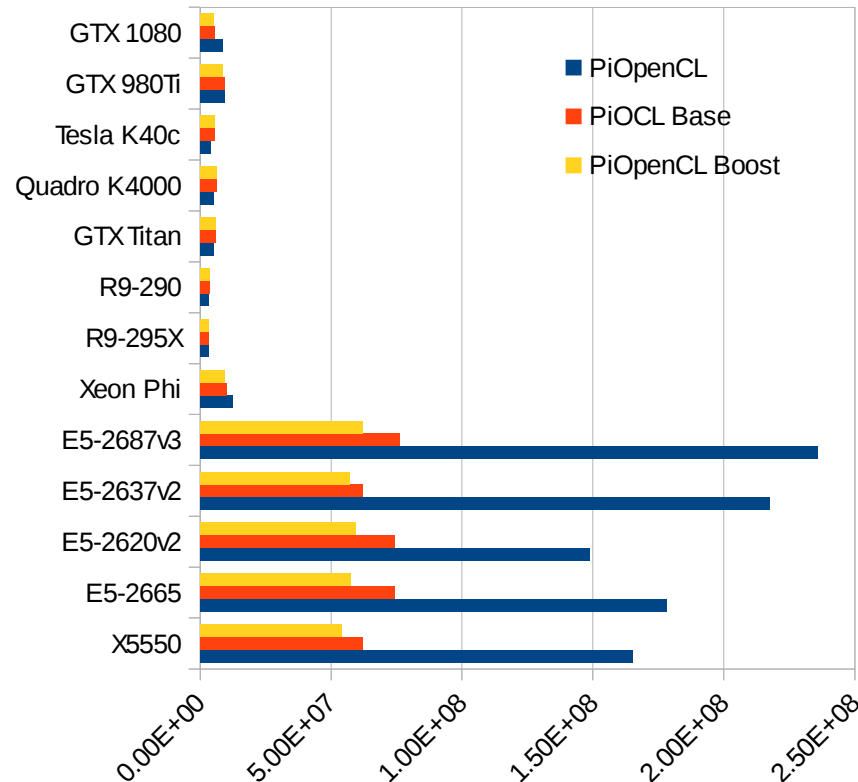
- 3 implémentations pour CPU:
 - Celle d'AMD: l'historique ...
 - Intel's: très efficace (mais pas toujours)
 - Celle OpenSource PortableCL (POCL)
- 4 implémentations pour GPU:
 - Celle d'AMD : pour ses cartes...
 - Celle de Nvidia: pour les circuits éponymes
 - "Beignet": version Open Source pour les circuits Intel (très récents)
 - Mesa: version Open Source pour les circuits récents mais pas trop...
- 1 implémentation pour Accélérateur:
 - Intel's: for the MIC Xeon Phi Knight Corner
- 1 implémentation pour FPGA :
 - Celle d'Altera notamment (racheté par Intel)
- Pour les autres plates-formes, possible (possible mais pas la moindre expérience)

Après le CPU et le GPU, Définition du QPU et de l'EPU

- PR: "Parallel Rate" or « Régime de parallélisme »
- CPU: Central Processing Unit
- GPU: Graphical Processing Unit
- QPU: Quantum Processing Unit
 - Pour un usage, on lance avec $PR = 1$
 - Exemple: lancement de Threads = 1, Blocks = 1 ou Work Items = 1
- EPU: Equivalent Processing Unit
 - Pour un usage, performance optimale obtenue pour un certain PR
 - Exemple: pour une GTX1080Ti, maximum pour 16x le nombre de « cudacores »

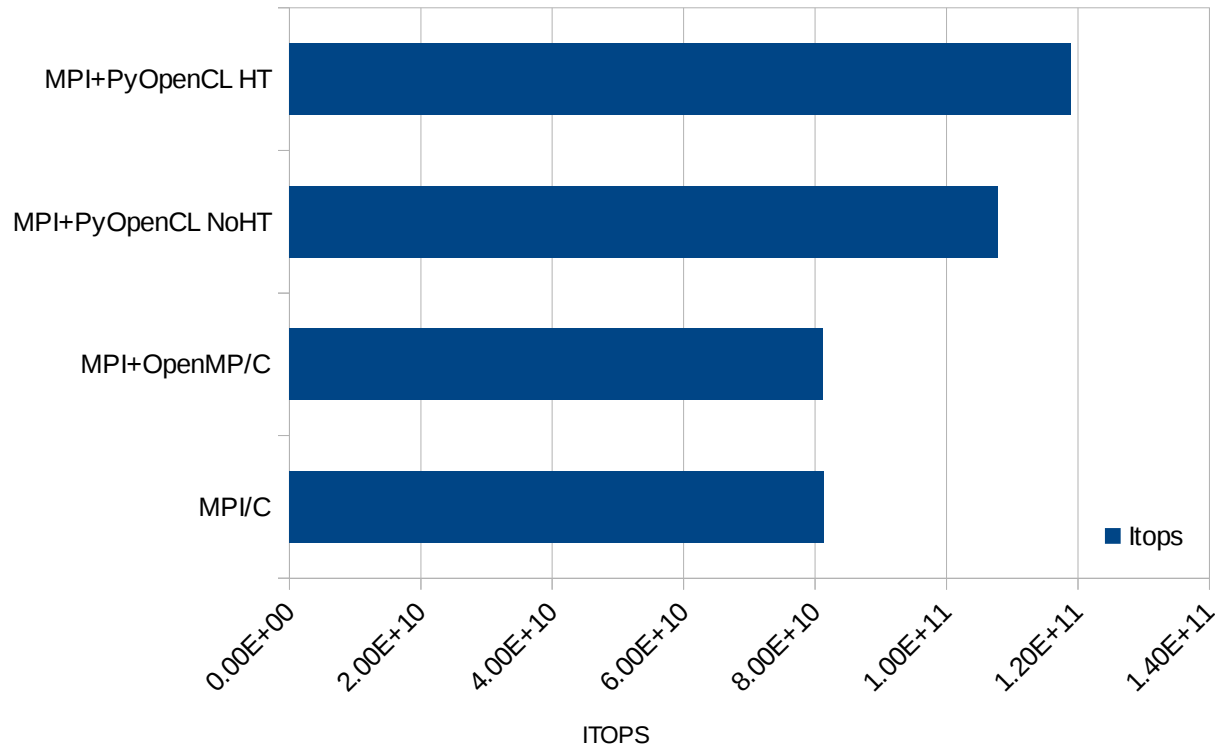
Pour 1 de QPU, « bas régime » ... Le CPU explose le GPU!

- De 20 à 50x plus lent !



- 1.5 ordre de magnitude

Pour préparer la prochaine... Et Python dans tout ça ?

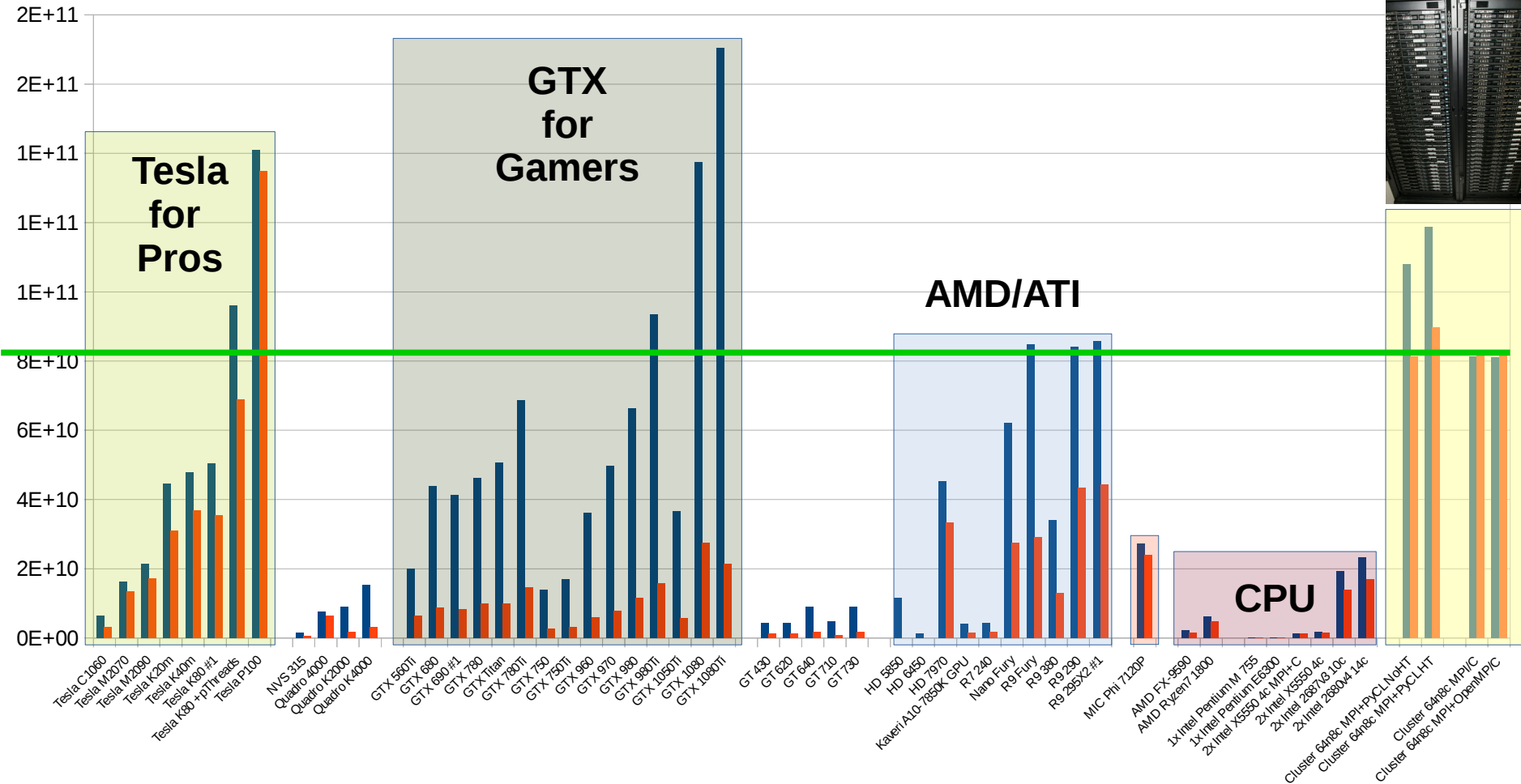


- 64 nœuds avec 512 cœurs
- 3 implémentations (2 hybrides)



Pi Monte Carlo: le match ...

Python vs C & CPU vs GPU vs Phi



Et que faut-il en retenir ?

Caractériser pour éviter les regrets

- Ce qu'il faut retenir :
 - Dans une machine standard, en 2018, la « puissance brute » est dans le GPU
 - Pour un parallélisme faible, le CPU est beaucoup plus efficace que le GPU
 - Le GPU l'emporte seulement si le régime de parallélisme est supérieur à 1000
 - Le GPU ne délivre sa puissance optimale « que » pour certains PR
 - Sans caractérisation systématique, des déconvenues sont à prévoir
 - OpenCL est le meilleur compromis pour programmer les *PU
 - Python reste la meilleure approche pour programmer OpenCL
- Que faut-il faire ? Expérimenter !