

Du « multi-cœurs » des processeurs au « myri-alus » des GPUs :

“La” technologie “disruptive”
vue par un physicien

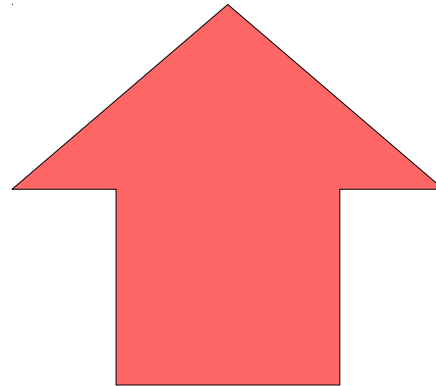
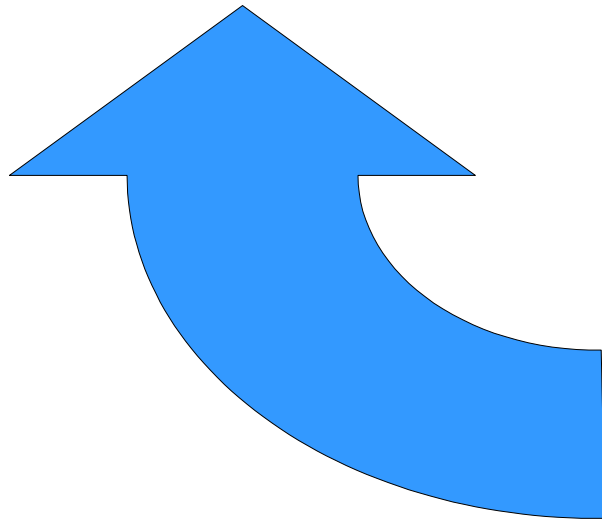
Emmanuel Quémener

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

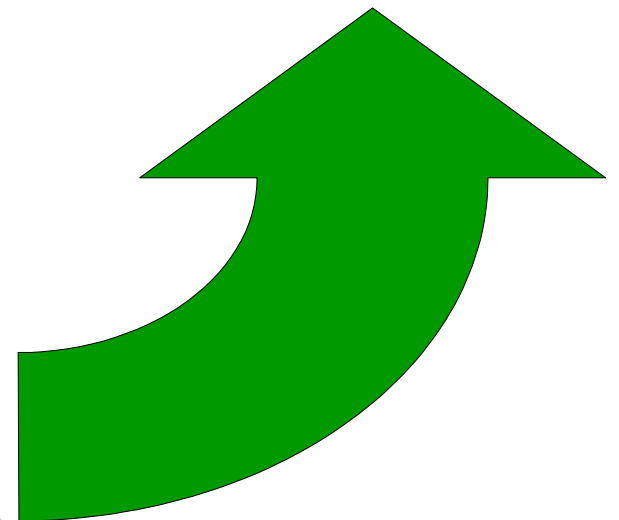
Myriade : “nombre de dix mille”

Le Centre Blaise Pascal « Maison de la modélisation » et ... Conférences

Formations



Projets



Hôtel

CBP

CENTRE BLAISE PASCAL



Centre Blaise Pascal

Plateforme expérimentale avec ses 10 plateaux techniques

- **Multi-nœuds** : 5 clusters de 4 à 64 nœuds, Nœuds/Cœurs : 4/24, 8/64, 8/64, 64/512
- **Multi-cœurs** : 10 de 2 à 20 cœurs,
 - Nœuds/Serveurs : de 8 à 20 cœurs, Stations de travail : de 2 à 16 cores, 10 types de CPU différents
- **GPU & Accélérateur** : 36 modèles différents de GPU (AMD & Nvidia), 1 Intel MIC
 - GPGPU : 8 ; GPU Nvidia : 18 ; GPU AMD/ATI : 10 types ; Xeon Phi 7120P
- **Intégration** : 16 machines : Debian de Lenny à Sid en 32 & 64 bits, ...
- **Matériel exotique** : 3 machines ARMv7 sous Debian Jessie ou Ubuntu
- **Visualisation 3D** : 2 stations , 2 vidéoprojecteurs, 20 moniteurs, 4 paires de lunettes
- **Bureau à distance** : plus de 25 hôtes avec le couple x2go/VirtualGL
- **COMOD** avec SIDUS : « Compute On My Own Device » pour les utilisateurs des laboratoires
 - SIDUS est « Single Instance Distributing Universal System »
- **Démonstrateur Galaxy** pour le traitement intensif de données biologiques

Centre Blaise Pascal ~ Dryden FR

Un petit exemple illustratif



Dryden Flight Research Center EC87 0182-14 Photographed 1987
X-29

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

Recycle, Réutilise et explore de nouveaux domaines

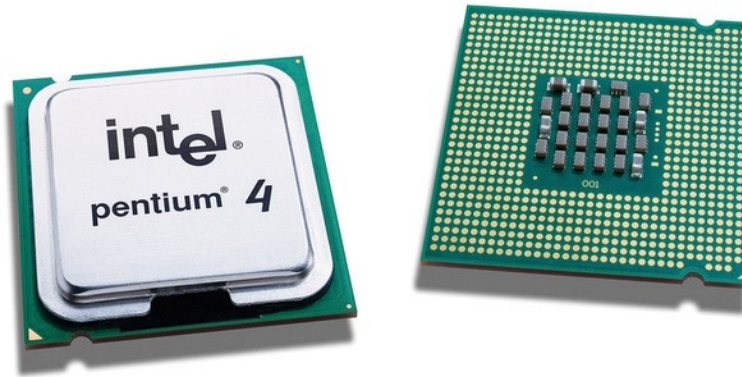
Quel regard sur la discipline ?

Celui d'un physicien

- Approche « système » du physicien
 - Héritage des calculateurs analogiques
 - Le « système » informatique :
 - réseau/matériel/OS/logiciel/codes/usages/...
- Approche « Saint Thomas »
 - Appréhension par sa seule mesure
- Approche « pilote d'essais »
 - Caractérisation, recherche d'une exploitation optimale...

L'énergie en informatique...

Le point de vue du physicien

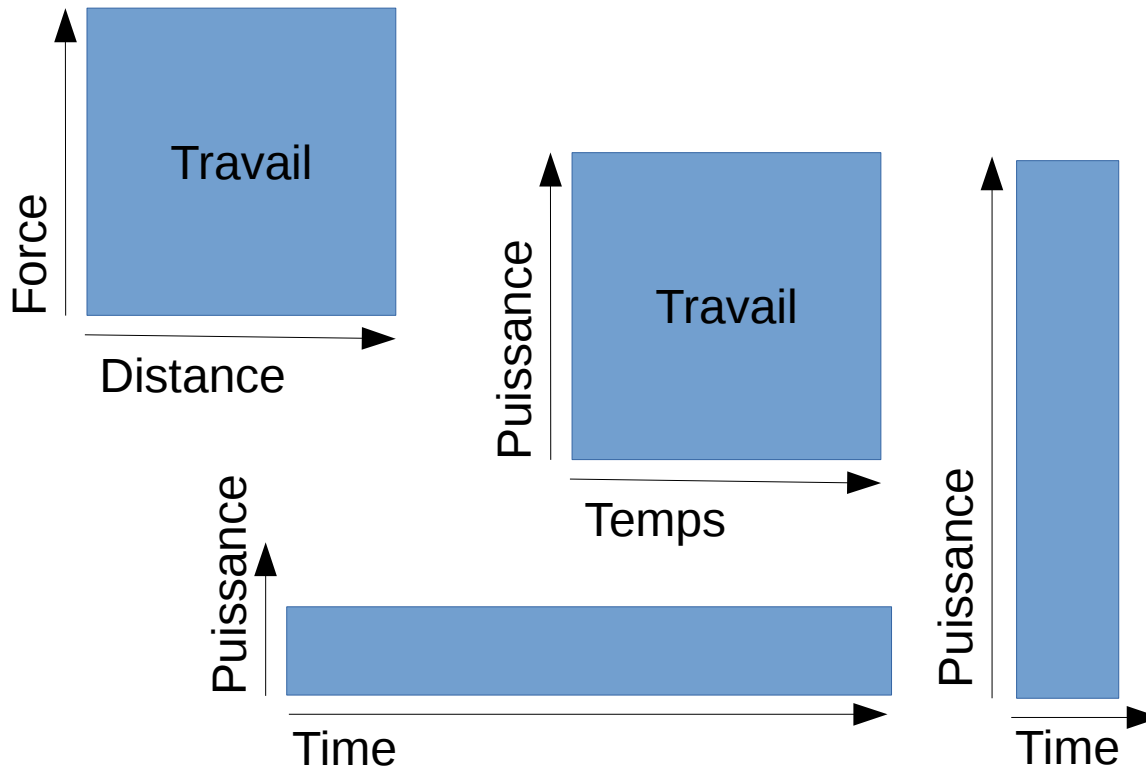


Électronique & Thermique



L'énergie en informatique...

Le point de vue du physicien



Mécanique

La puissance comme le produit de

- La fréquence
- Nombre d'unités
- Puissance d'une unité

$$W = \int_{x_1}^{x_2} F dx = \int_{t_1}^{t_2} P dt$$

Caractériser la puissance de « moteurs » informatiques

Longtemps, entre 1985 et 2005,la variable était la fréquence !

Chart 5: Horsepower curves for 5 different motorcycles

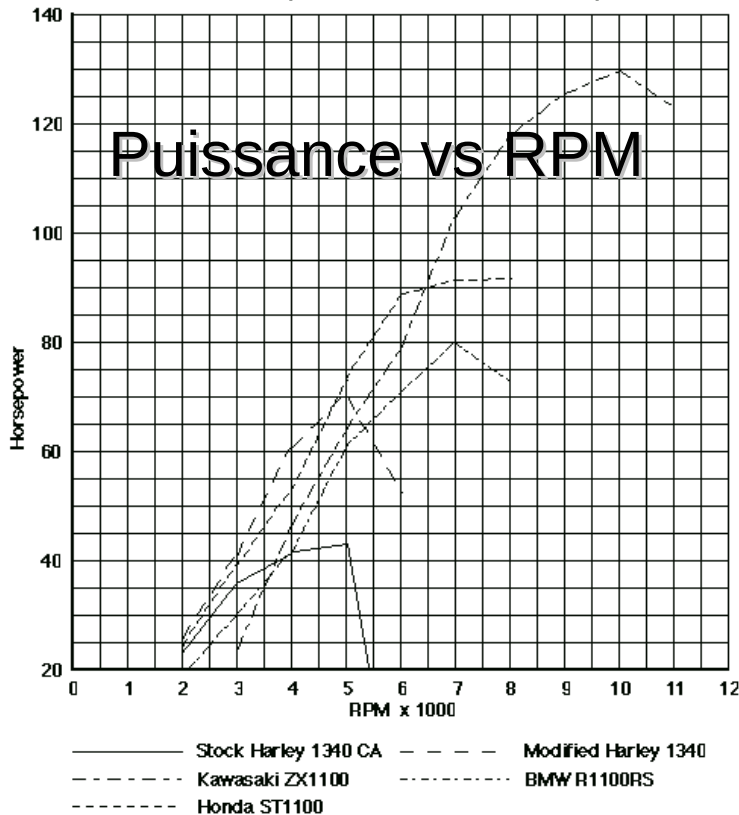
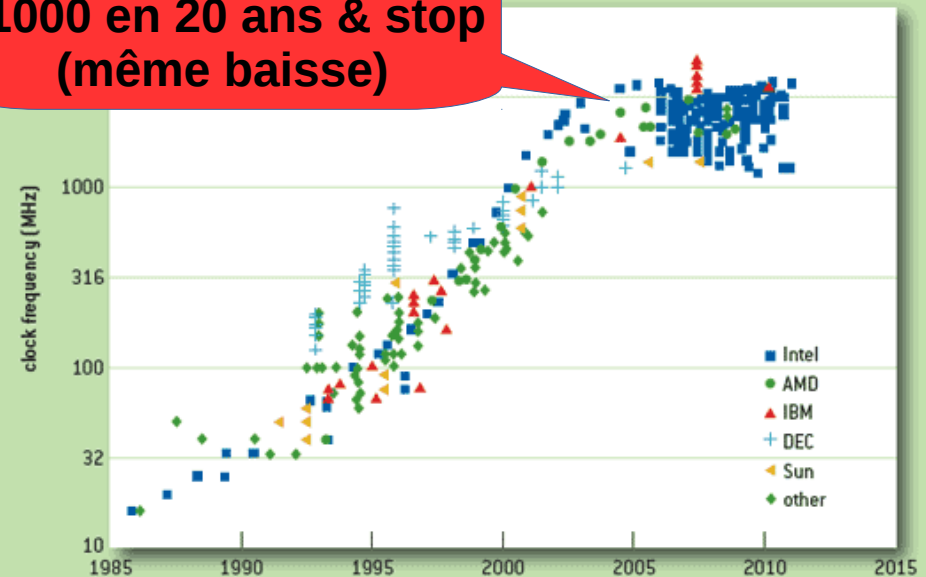


FIGURE 7

Processor Frequency Scaling Over Time

x1000 en 20 ans & stop
(même baisse)



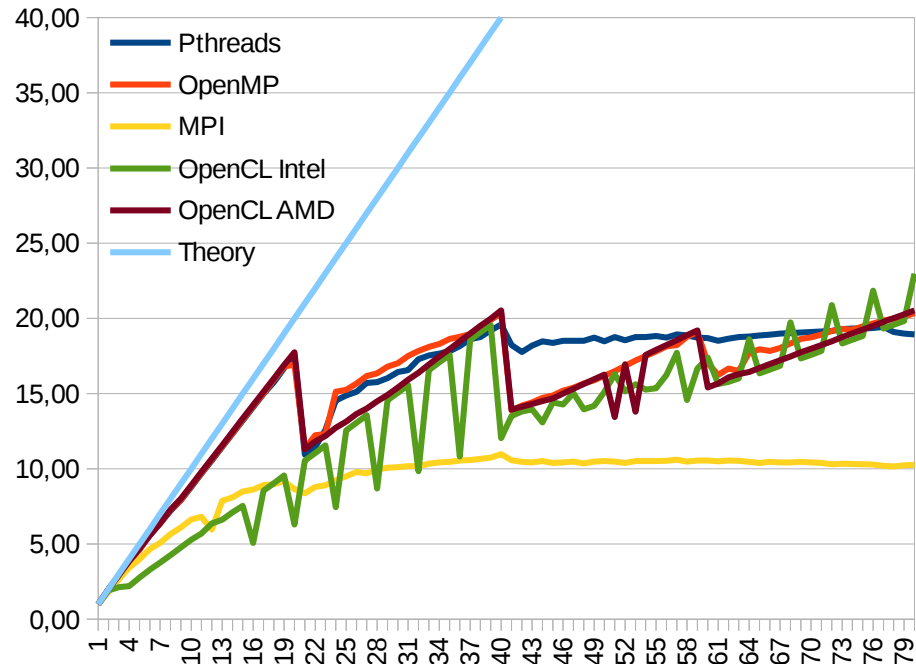
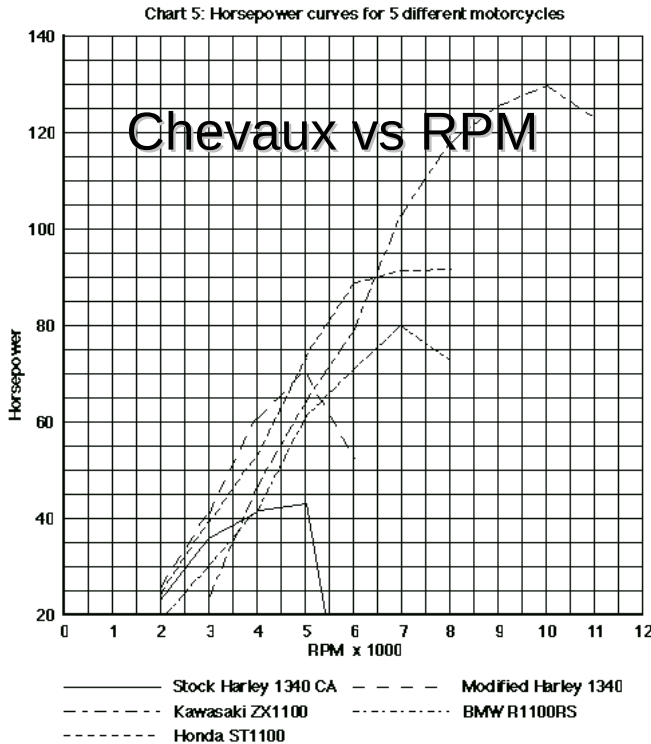
Pourquoi la multiplication des PU ?

Contourner le « mur » de la TDP...

- Grandeur & décadence de la fréquence
 - Entre 1989 et 1999 : de 4 MHz à 400 MHz x100 en 10 ans
 - Entre 1999 et 2004 : de 400 MHz à 3 GHz x~10 en 5 ans
 - Entre 2004 et 2009 : de 3 GHz à 2 GHz
- TDP : *Thermal Design Power* : puissance limite du socket
 - $TDP = \frac{1}{2} C V^2 f$
 - C = Capacitance, f = fréquence, V = tension, mais **V=fonction(fréquence)**
 - TDP (ou enveloppe thermique) pour un socket : 150 W (sur 4 cm²)
 - Capacitance = Finesse ² . Nb Transistors . Constante de Mylq (~ 0.015)
- **Solution viable : nombre d'unités de puissance (PU)**

Le « moteur », source de « puissance »

Quitter la fréquence, changer RPM en PR

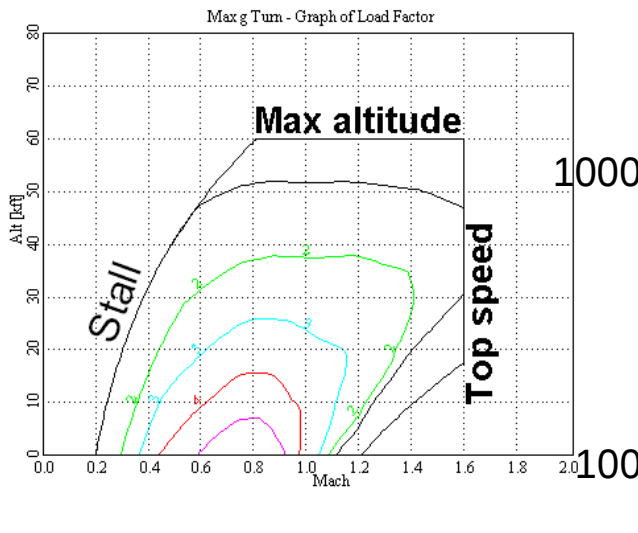


- Quels « comportements » pour ces moteurs à la « charge » ?
- Le « moteur », un système intriquant matériel, OS et logiciel

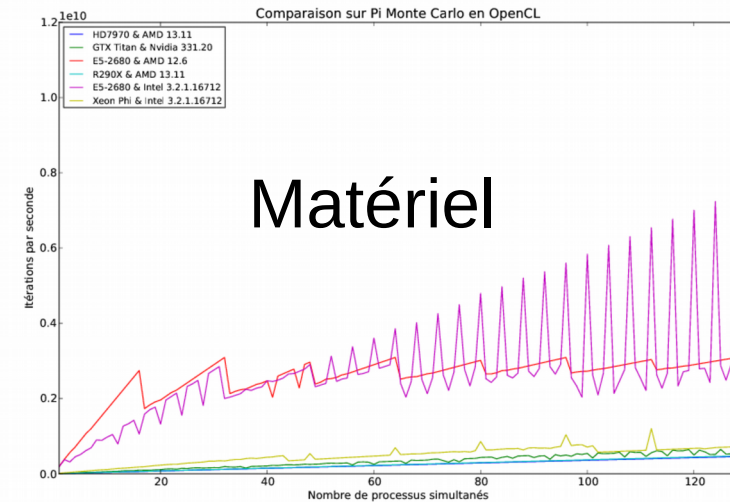
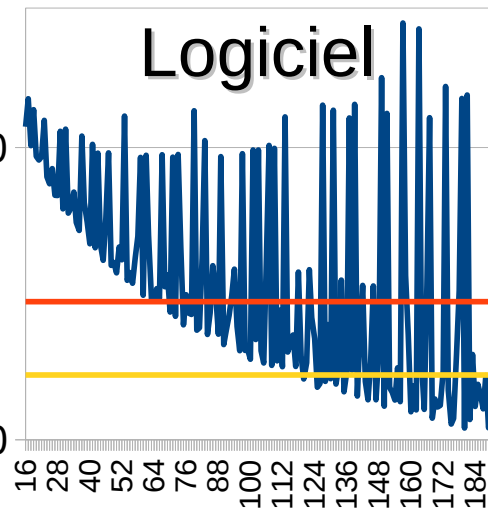
De moteurs à enveloppes de //isme

Une estimation nécessaire ?

Enveloppe de vol



Enveloppe de parallélisation

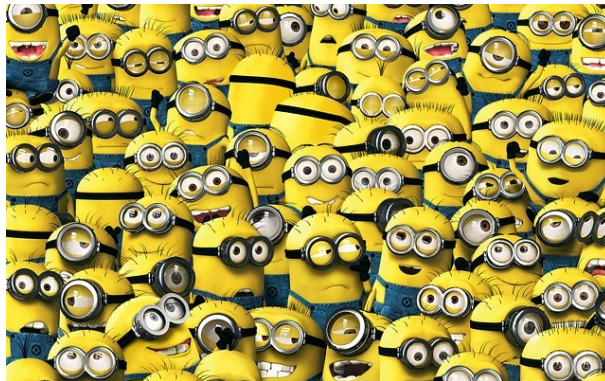


- Vitesse/altitude
- facteur de charge
- Parallélisation/Mémoire/CPU/GPU
- Entrées/Sorties/Contextes/...

Des (multi|many)-cores aux myri-alus ?

Un laboratoire de parallélisme

- Vous avez de 1 à des dizaines de milliers de PU équivalents !
- Vous avez le choix entre : **CPU MIC GPU**

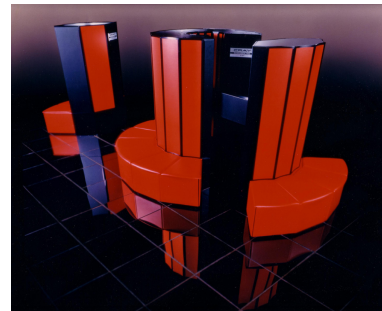


Comment les choisir & les distinguer pour son usage ?

Une génération (humaine)...

Un film de série B en 1984

- 1984 : The Last Starfighter
 - 27 minutes d'image de synthèse
 - $22.5 \cdot 10^9$ opérations par image
 - Un Cray X-MP (130 kW)
 - 66 jours (en fait, 1 année nécessaire)
- 2016: sur la GTX 1080Ti (180 W)
 - 3.4 secondes
 - Comparaison GTX1080Ti / Cray
 - Performance : $\times 1600000$!
 - Consommation / 9000000000 !

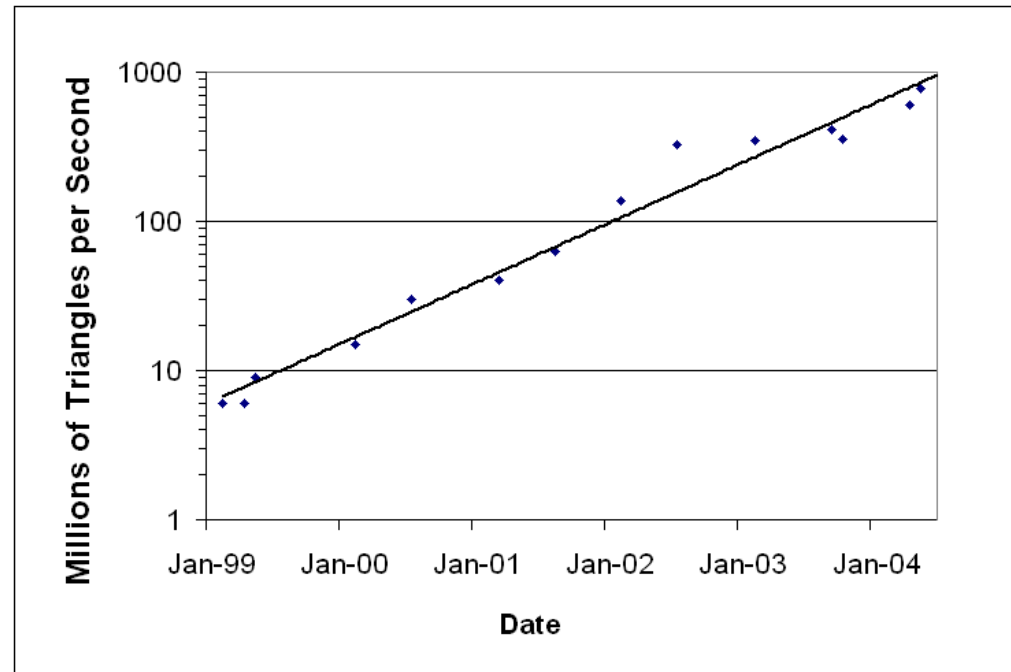


Pourquoi le GPU est « disruptif » ?

Le « grand détournement » en HPC

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

- x100 en 5 ans



- Entre 2000 et 2015

- GeForce 2 Go/GTX 980Ti : de 286 à 2816000 MOpérations/s : x10000
 - Pour un CPU « classique » : x100

Au commencement de toute chose

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



Position du GPU face aux autres?

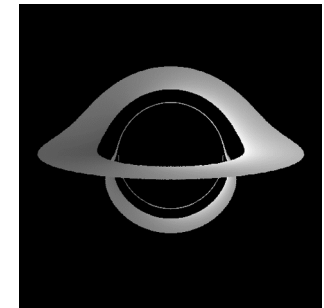
Les autres « accélérateurs »

- Accélérateur ou la vieille histoire des coprocesseurs...
 - 1980 : 8087 (sur 8086/8088) pour les opérations en virgule flottantes
 - 1989: 80387 (sur 80386) et son respect du IEEE 754
 - 1990 : 80486DX et l'intégration du FPU dans le CPU
 - 1997 : K6-3DNow ! & Pentium MMX : SIMD dans le CPU
 - 1999 : fonctions SSE et le début d'une longue série (SSE4 & AVX)
- Quand les circuits restent hors du CPU
 - 1998 : Les DSP de la catégorie des TMS320C67x comme outils
 - 2008: le Cell dans la PS3, IBM dans le Road Runner & un Top1 au Top500
- Travail de compilateurs & forte dépendance au modèle

Pourquoi le GPU est-il si puissant ?

Pour construire une image 3D !

- 2 approches:
 - *Raytracing* : PovRay (« dimensions » de l'UMPA)
 - *Shading* : 3 opérations
- « Raytracing » :
 - De l'oeil vers les contacts de chaque objet
- « Shading »
 - Model2World : objets vectoriels placés dans la scène
 - World2View : projection des objets dans un plan de vue vectoriel
 - View2Projection : pixellisation du plan de vue

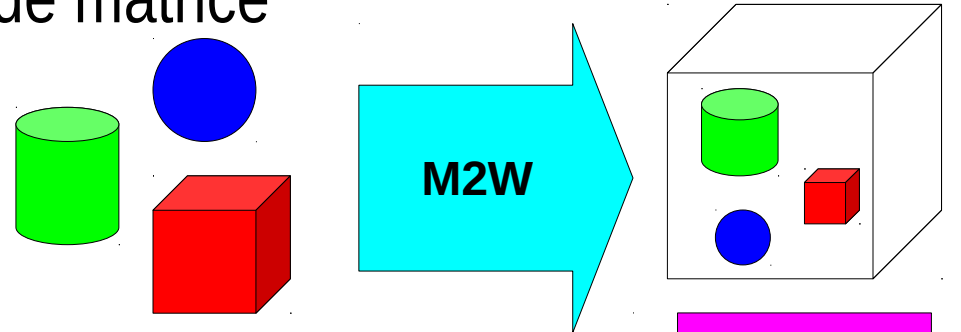


Pourquoi le GPU est-il si puissant ?

Shadering & Calcul matriciel

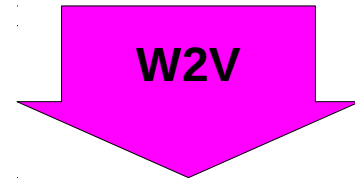
- **Model 2 World** : 3 produits de matrice

- Rotation
- Translation
- Mise à l'échelle



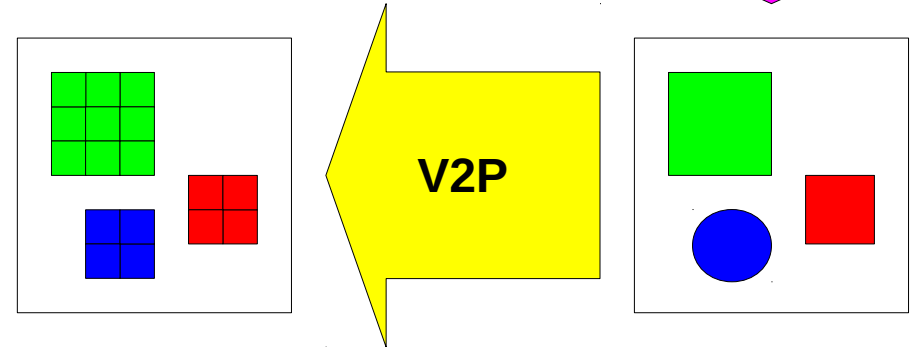
- **World 2 View** : 2 produits de matrice

- Positionnement de la caméra
- Direction de l'endroit pointé



- **View 2 Projection**

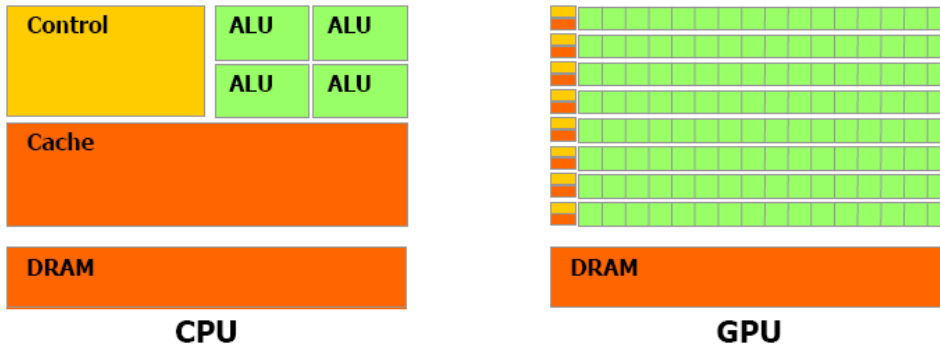
- Pixellisation



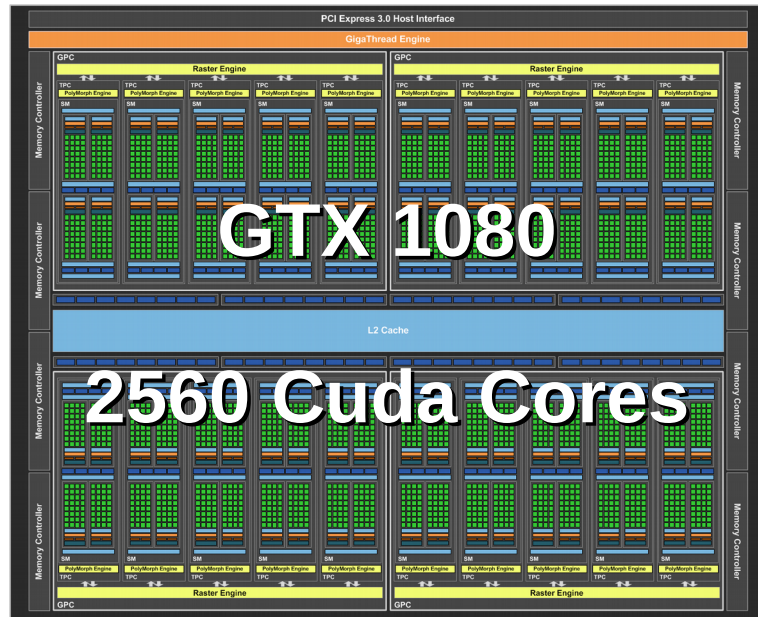
Donc, un GPU : c'est un « gros » Multiplicateur de Matrices

Combien de composants à l'intérieur ?

Quelle différence entre GPU & CPU



- Opérations
 - Multiplication de Matrice
 - Vectorisation
 - « Pipelining »
 - Shader (multi)processeur
- Programmation : 1993
 - OpenGL, Glide, Direct3D,
- Généricité : 2002
 - CgToolkit, CUDA, OpenCL



Pourquoi le GPU est-il si puissant ?

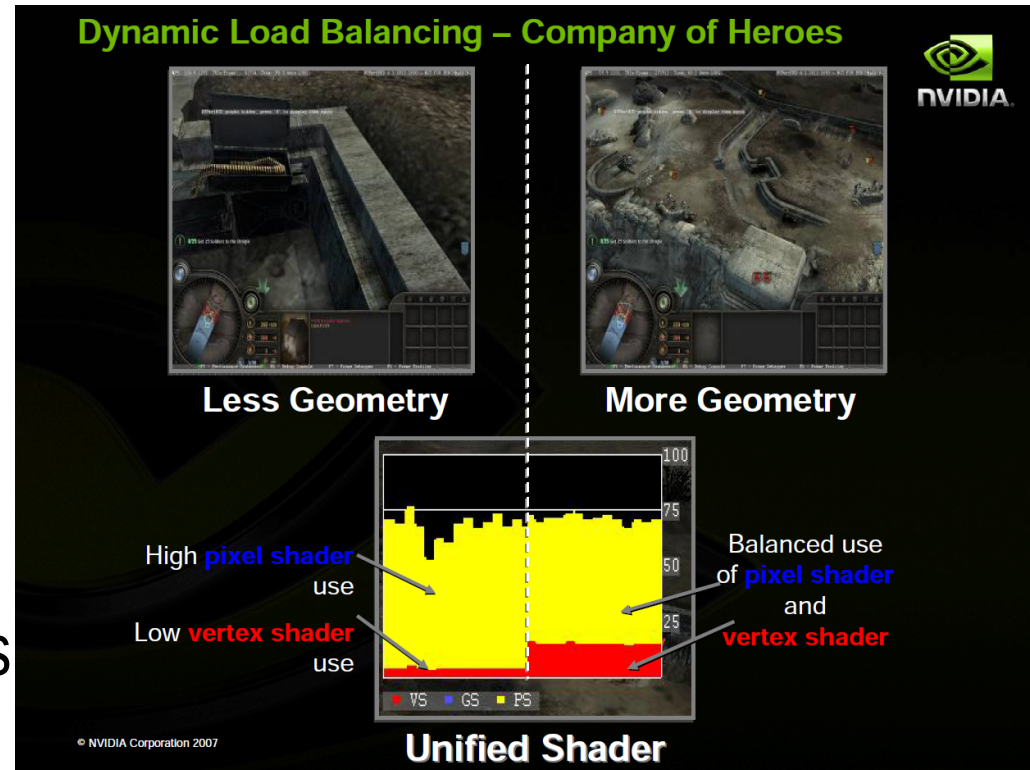
Le retour à une base « générique »

Dans le GPU

- Unités spécialisées
- Efficacité du pipelining

Perte d'adaptativité

- Scènes changeantes
- Nature des détails différentes



La solution

Le retour à des unités de traitements plus génériques !

Pourquoi le GPU est-il si puissant ?

Tous les « M » de la taxonomy de Flynn

- Vectoriel: SIMD (Simple Instruction Multiple Data)
 - Addition de 2 positions (x,y,z) : 1 commande unique
- Parallèle : MIMD (Multiple Instructions Multiple Data)
 - Plusieurs exécutions indépendantes avec (à peu près) les mêmes données
- En fait, SIMT : Simple Instruction Multiple Threads
 - Toutes les unités de traitement partagent les Threads
 - Chaque traitement peut être indépendant des autres
- Nécessité pour les « Threads » d'être synchronisés...

Petite frise chronologique...

(sur comment les « dompter »)

- 1992-01 : OpenGL et la naissance d'un standard (pas trop ouvert)
- 1998-03 : OpenGL 1.2 et fonctions intéressantes
- 2002-12 : Cg Toolkit (Nvidia) et des extensions de langage
 - Wrappers pour tous les langages (dont Python :-)
- 2007-06 : CUDA (Nvidia) ou l'arrivée d'un langage réel
- 2008-06 : Snow Leopard (Apple) intègre OpenCL
 - La volonté d'utiliser au mieux sa machine ?
- **2008-11 : OpenCL 1.0 et ses premières spécifications**
- 2011-04 : WebCL et sa première version par Nokia

Qui utilise OpenCL ?

Du côté des applications...

- OS : Apple dans MacOSX
- Applications « gargantuesques » :
 - Libreoffice
 - Ffmpeg
- Applications « métiers »
 - Les graphiques : photoscan, MorphoGraphX
 - Les scientifiques : OpenMM, Gromacs, Lammmps, Amber, AceMD, ...
- Des centaines de logiciels & librairies ! Mais vérifier est essentiel...
 - <https://www.khronos.org/opencl/resources/opencl-applications-using-opencl>

Petit tour d'horizon du parallélisme

Modèle de programmation parallèle

	Cluster	Nœud CPU	Nœud GPU	Nœud Nvidia	Accélérateur	FPGA
MPI	Oui	Oui	Non	Non	Oui*	Non
PVM	Oui	Oui	Non	Non	Oui*	Non
OpenMP	Non	Oui	Non	Non	Oui*	Non
Pthreads	Non	Oui	Non	Non	Oui*	Non
OpenCL	Non	Oui	Oui	Oui	Oui	Oui
CUDA	Non	Non	Non	Oui	Non	Non
TBB	Non	Oui	Non	Non	Oui*	Non

- OpenCL « semble » dont le plus polyvalent :-)
- CUDA peut SEULEMENT être utilisé avec Nvidia
- Les accélérateurs semblent les plus universels, mais...

Agir en « intégrateur » : Ne pas réinventer la roue...

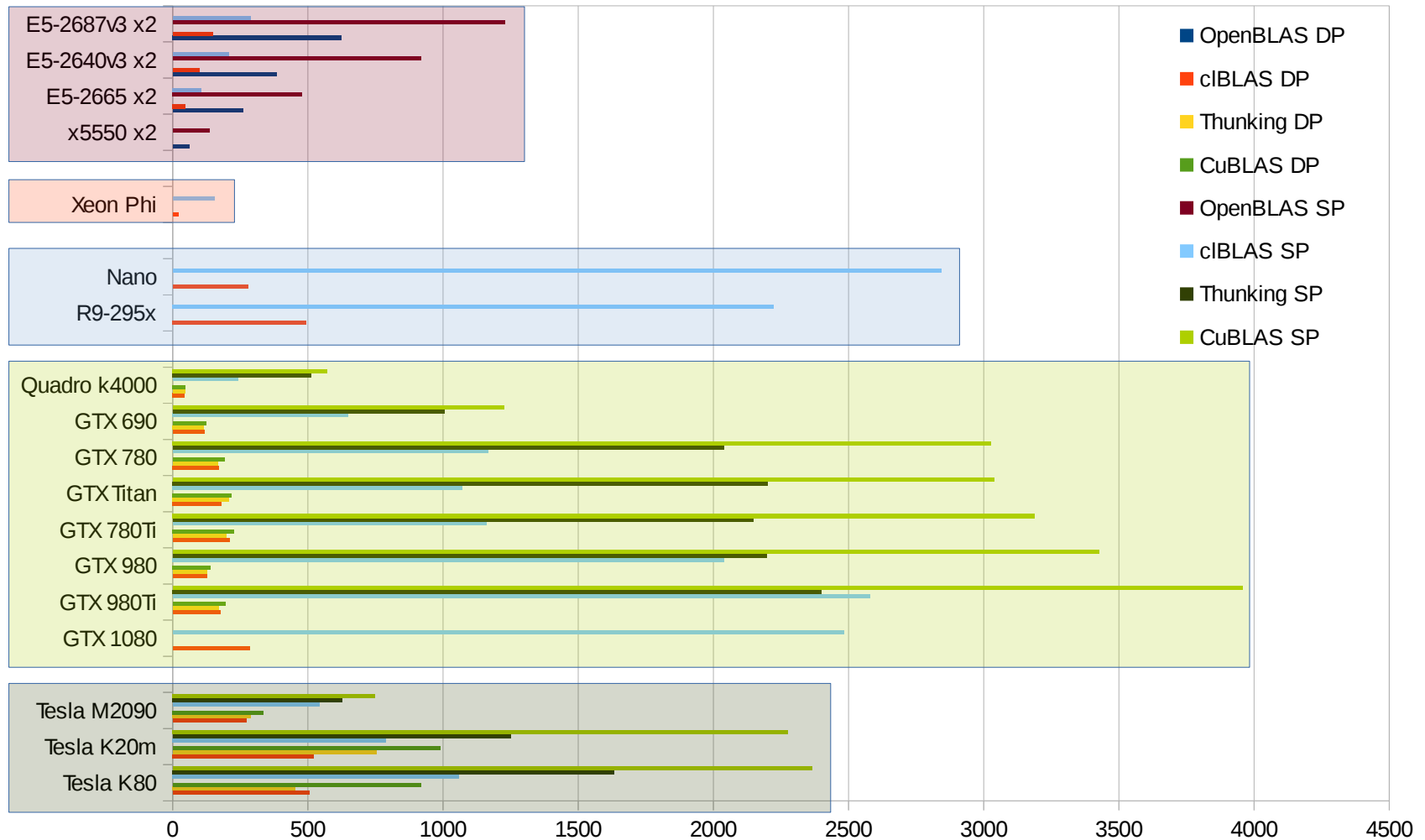
Librairies de programmation parallèle

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

- Les librairies classiques utilisables avec les GPU
 - Nvidia fournit plein d'implémentations (notamment BLAS)
 - OpenCL fournit certaines, mais pas complète !

BLAS comparison : CPU vs GPU

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

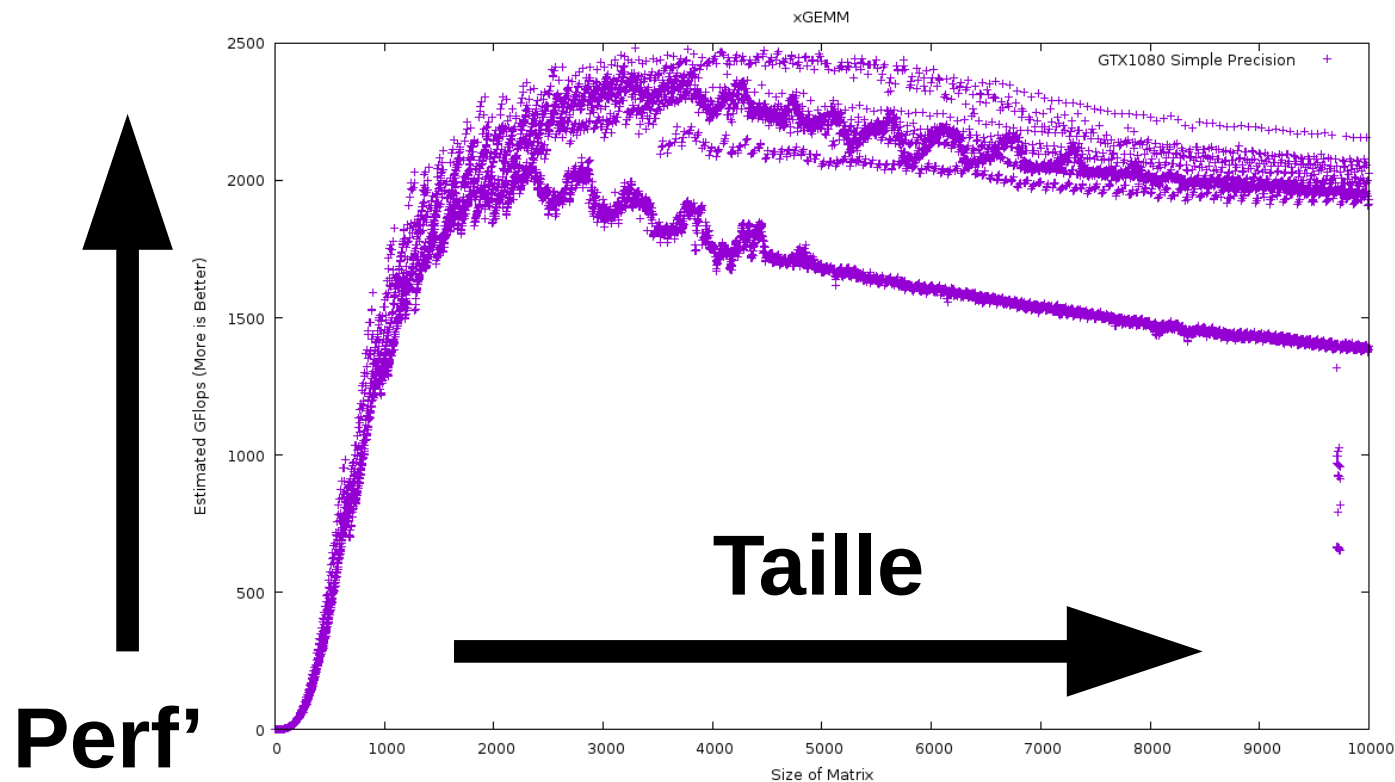


Déjà en 2010, étrange...

Le comportement Tesla C1060...

- Uniquement produit matriciel : xGEMM
 - Propriété : Transposé $(A * B) = \text{Transposé}(A) * \text{Transposé}(B)$
- Résultats (en Gflops) : Yesss !
 - SP : FBLAS/CBLAS : 12, CuBLAS : 350/327 : x27 !!!
 - DP : FBLAS/CBLAS : 6, CuBLAS : 73/70 : x11 !
- Surprise : Ah !!! CuBLAS préfère les x16 !
 - SP : 16000^2 , 350, mais 15999^2 ou 16001^2 , 97 : x3,6 !
 - DP : 10000^2 , 73, mais 9999^2 ou 10001^2 , 31 : x2,35

Ce qu'on ne diffuse jamais... xGEMM sur un domaine...



Bref, la performance est là, mais pas partout...

Pourquoi OpenCL ?

Une histoire politiquement très incorrecte

- Au milieu des années 2000, MacOSX exige plus de puissance
 - Traitements « gérables » par un chipset Nvidia embarqué
 - CUDA existe comme successeur de CgToolkit
- Mais Apple ne veut pas être « prisonnier »
 - (comme ses utilisateurs ;-)
- Apple participe à la création du consortium Khronos
 - AMD, IBM et ARM s'y collent
 - ... puis Intel et Nvidia y viennent à reculons...

Qu'est-ce que OpenCL offre 10 implémentations on x86

- 3 implémentations pour CPU :
 - Celle de AMD : la première...
 - Celle de Intel : très efficace (mais pas toujours)
 - Celle OpenSource PortableCL (POCL)
- 4 implémentations pour GPU :
 - Celle propriétaire de AMD : pour les circuits graphiques AMD/ATI
 - Celle de Nvidia : pour les circuits éponymes
 - Celle « Beignet » : Open Source pour les circuits graphiques Intel **très** récents
 - Celle Mesa : Open Source pour les circuits graphiques avec des bons supports noyaux
- 1 implementation pour Accélérateur :
 - Celle d'Intel : pour le MIC Xeon Phi Knight Corner
- Pour les autres plateformes, possible (mais pas expérimenté)

But ultime dans la programmation de base : remplacer les boucles par des distributions

- Blocks/WorkItems
 - Domaine « global », mémoire accessible étendue mais lente...
- Threads
 - Domaine « local », mémoire accessible limitée mais rapide
 - Nécessité de synchroniser les processus
- Localité importante : différents accès à la mémoire...
- « clinfo » pour les paramètres : Max work items
 - Max work items : $1024 \times 1024 \times 1024$ pour un CPU donc distribution de 10^9 !

Comment ?

Un « Hello World » en OpenCL...

- Définir deux vecteurs en ASCII
- Dupliquer les vecteurs en 2 gros
- Ajouter les vecteurs en OpenCL
- Récupérer de résultat
- L'imprimer

Addition de 2 vecteurs $a+b=c$

Pour tout n :

$$c[n] = a[n] + b[n]$$

[illegible]

Code Kernel : Construction & Appel

Add a vector in OpenCL

```
__kernel void VectorAdd(__global int* c, __global int* a, __global int* b)
{
    // Index of the elements to add
    unsigned int n = get_global_id(0);
    // Sum the n th element of vectors a and b and store in c
    c[n] = a[n] + b[n];
}
```

```
OpenCLProgram = cl.Program(ctx, OpenCLSource).build()
```

```
OpenCLProgram.VectorAdd(queue, HostVector1.shape,
None, GPUOutputVector , GPUVector1, GPUVector2)
```

Comment faire en OpenCL ? Écrire « Hello World ! » en C

```
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>

const char* OpenCLSource[] = {
    "__kernel void VectorAdd(__global int* c, __global int* a, __global int* b)",
    "{",
    "    // Index of the elements to add \n",
    "    unsigned int n = get_global_id(0);",
    "    // Sum the n'th element of vectors a and b and store in c \n",
    "    c[n] = a[n] + b[n];",
    "}"
};

int InitialData1[20] = {37,50,54,50,56,0,43,48,74,71,32,36,16,43,56,100,50,25,15,17};
int InitialData2[20] = {35,51,54,58,55,32,36,69,27,39,55,40,16,44,55,14,58,75,18,15};
#define SIZE 2048

int main(int argc, char **argv)
{
    int HostVector1[SIZE], HostVector2[SIZE];
    for(int c = 0; c < SIZE; c++)
    {
        HostVector1[c] = InitialData1[c%20];
        HostVector2[c] = InitialData2[c%20];
    }

    cl_platform_id cpPlatform;
    clGetPlatformIDs(1, &cpPlatform, NULL);
    cl_int ciErr1;
    cl_device_id cdDevice;
    ciErr1 = clGetDeviceIDs(cpPlatform, CL_DEVICE_TYPE_GPU, 1, &cdDevice, NULL);
    cl_context GPUContext = clCreateContext(0, 1, &cdDevice, NULL, NULL, &ciErr1);
    cl_command_queue cqCommandQueue = clCreateCommandQueue(GPUContext,
    cdDevice, 0, NULL);
```

Kernel OpenCL

```
    cl_mem GPUVector1 = clCreateBuffer(GPUContext, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(int) *
    SIZE, HostVector1, NULL);
    cl_mem GPUVector2 = clCreateBuffer(GPUContext, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, sizeof(int) *
    SIZE, HostVector2, NULL);
    cl_mem GPUOutputVector = clCreateBuffer(GPUContext, CL_MEM_WRITE_ONLY, sizeof(int) * SIZE, NULL,
    NULL);
    cl_program OpenCLProgram = clCreateProgramWithSource(GPUContext, 7, OpenCLSource, NULL, NULL);
    clBuildProgram(OpenCLProgram, 0, NULL, NULL, NULL, NULL);
    cl_kernel OpenCLVectorAdd = clCreateKernel(OpenCLProgram, "VectorAdd", NULL);
    clSetKernelArg(OpenCLVectorAdd, 0, sizeof(cl_mem), (void*)&GPUOutputVector);
    clSetKernelArg(OpenCLVectorAdd, 1, sizeof(cl_mem), (void*)&GPUVector1);
    clSetKernelArg(OpenCLVectorAdd, 2, sizeof(cl_mem), (void*)&GPUVector2;
    size_t WorkSize[1] = {SIZE}; // one dimensional Range
    clEnqueueNDRangeKernel(cqCommandQueue, OpenCLVectorAdd, 1, NULL, WorkSize, NULL, 0, NULL, NULL);
    int HostOutputVector[SIZE];
    clEnqueueReadBuffer(cqCommandQueue, GPUOutputVector, CL_TRUE, 0, SIZE * sizeof(int),
    HostOutputVector, 0, NULL, NULL);
    clReleaseKernel(OpenCLVectorAdd);
    clReleaseProgram(OpenCLProgram);
    clReleaseCommandQueue(cqCommandQueue);
    clReleaseContext(GPUContext);
    clReleaseMemObject(GPUVector1);
    clReleaseMemObject(GPUVector2);
    clReleaseMemObject(GPUOutputVector);
    for (int Rows = 0; Rows < (SIZE/20); Rows++) {
        printf("\t");
        for(int c = 0; c < 20; c++) {
            printf("%c", (char)HostOutputVector[Rows * 20 + c]);
        }
        printf("\n\nThe End\n\n");
        return 0;
    }
```

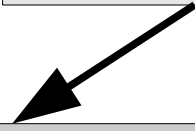
Appel Kernel

Nombre de lignes
Du kernel OpenCL

Comment faire en OpenCL ?

Écrire « Hello World ! » in Python

Kernel OpenCL



```
import pyopencl as cl
import numpy
import numpy.linalg as la
import sys

OpenCLSource = """
__kernel void VectorAdd(__global int* c, __global int* a, __global int* b)
{
    // Index of the elements to add
    unsigned int n = get_global_id(0);
    // Sum the n th element of vectors a and b and store in c
    c[n] = a[n] + b[n];
}
"""

InitialData1=[37,50,54,50,56,0,43,43,74,71,32,36,16,43,56,100,50,25,15,17]
InitialData2=[35,51,54,58,55,32,36,69,27,39,35,40,16,44,55,14,58,75,18,15]
SIZE=2048

HostVector1=numpy.zeros(SIZE).astype(numpy.int32)
HostVector2=numpy.zeros(SIZE).astype(numpy.int32)

for c in range(SIZE):
    HostVector1[c] = InitialData1[c%20]
    HostVector2[c] = InitialData2[c%20]

ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)
```

```
mf = cl.mem_flags

GPUVector1 = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR,
                        hostbuf=HostVector1)

GPUVector2 = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR,
                        hostbuf=HostVector2)

GPUOutputVector = cl.Buffer(ctx, mf.WRITE_ONLY,
                             HostVector1.nbytes)

OpenCLProgram = cl.Program(ctx, OpenCLSource).build()

OpenCLProgram.VectorAdd(queue, HostVector1.shape,
                        None, GPUOutputVector, GPUVector1, GPUVector2)

HostOutputVector = numpy.empty_like(HostVector1)

cl.enqueue_copy(queue, HostOutputVector, GPUOutputVector)

GPUVector1.release()
GPUVector2.release()
GPUOutputVector.release()

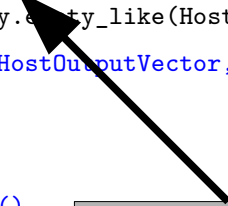
OutputString=''

for rows in range(SIZE/20):
    OutputString+='\t'
    for c in range(20):
        OutputString+=chr(HostOutputVector[rows*20+c])

print OutputString

sys.stdout.write("\nThe End\n\n");
```

Appel Kernel



Comment faire en OpenCL ?

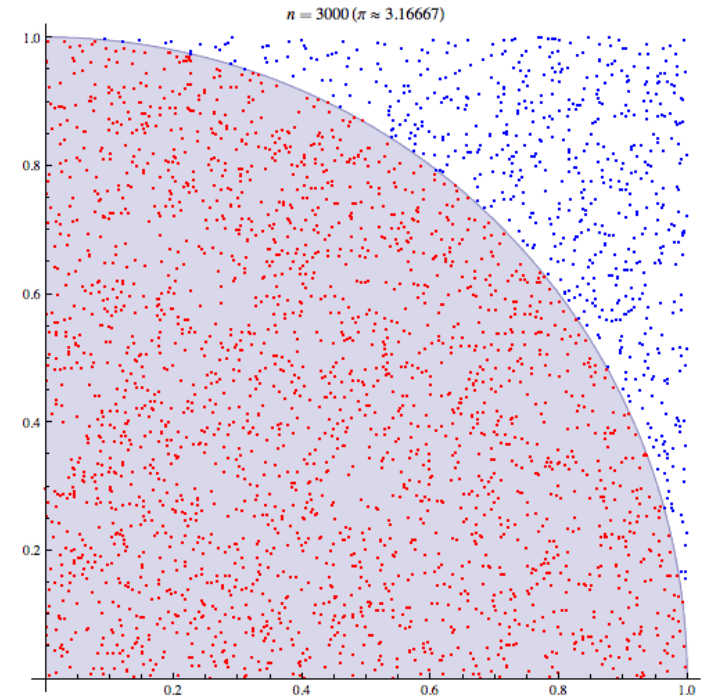
« Hello World » C/Python : la pesée

- Sur l'implémentation précédente en OpenCL :
 - En C : 75 lignes, 262 mots, 2848 octets
 - En Python : 51 lignes, 137 mots, 1551 octets
 - Facteurs : 0.68, 0.52, 0.54 pour les lignes, mots et octets...
- Programmation OpenCL :
 - Programmation des contextes pénible
 - « Ouvrir le carton d'IKEA est plus difficile que monter le meuble ! »
 - Nécessité de simplifier les appels par une API
 - Pas de compatibilité entre les SDK AMD et Nvidia
 - Chacun réécrit sa propre API
- Une solution simple et efficace, sinon « LA » solution : Python

Quel code simple à implémenter en // ?

PiMC : Pi par la méthode de la cible

- Exemple historique de la méthode de Monte Carlo
- Implémentaton parallèle :
 - Distribution équivalente des itérations
- De 2 à 4 paramètres
 - Nombre d'itérations
 - Régime de Parallélisme (Parallel Rate ou PR)
 - (Type de variable : INT32, INT64, FP32, FP64)
 - (RNG : MWC, CONG, SHR3, KISS)
- 2 observables simples :
 - Estimation de Pi (juste indicative, Pi n'étant PAS rationel :-))
 - Temps écoulé (individuel ou global)



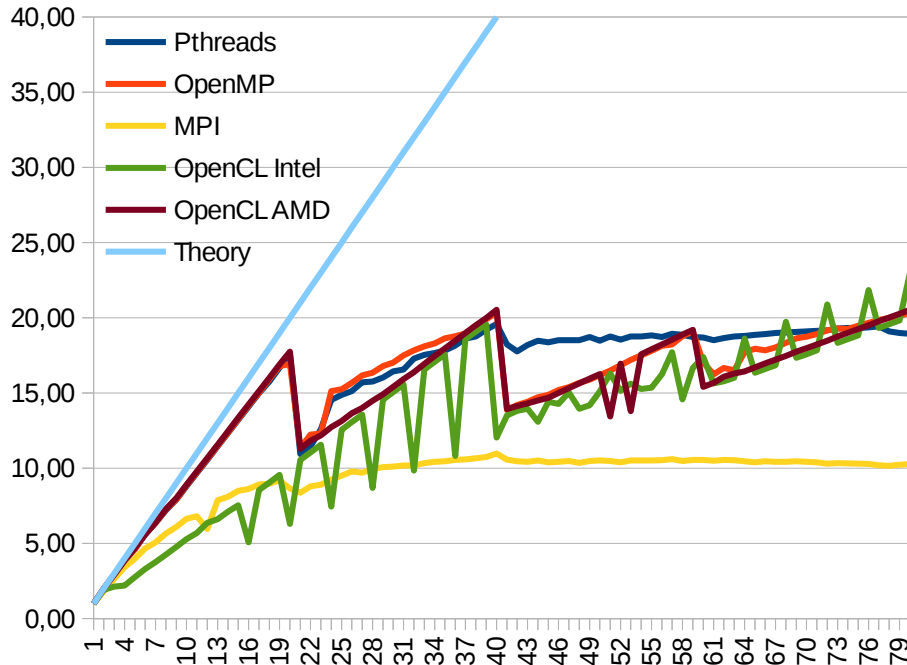
Différences entre CUDA/OpenCL

Les noyaux des calculs GPU

```
__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;
}
```

Des comportements étranges... sur un même processeur,



- Processeur 20 cœurs
 - E5-2670v2
- 5 implémentations
 - Dont 2 OpenCL...
- Des similitudes...

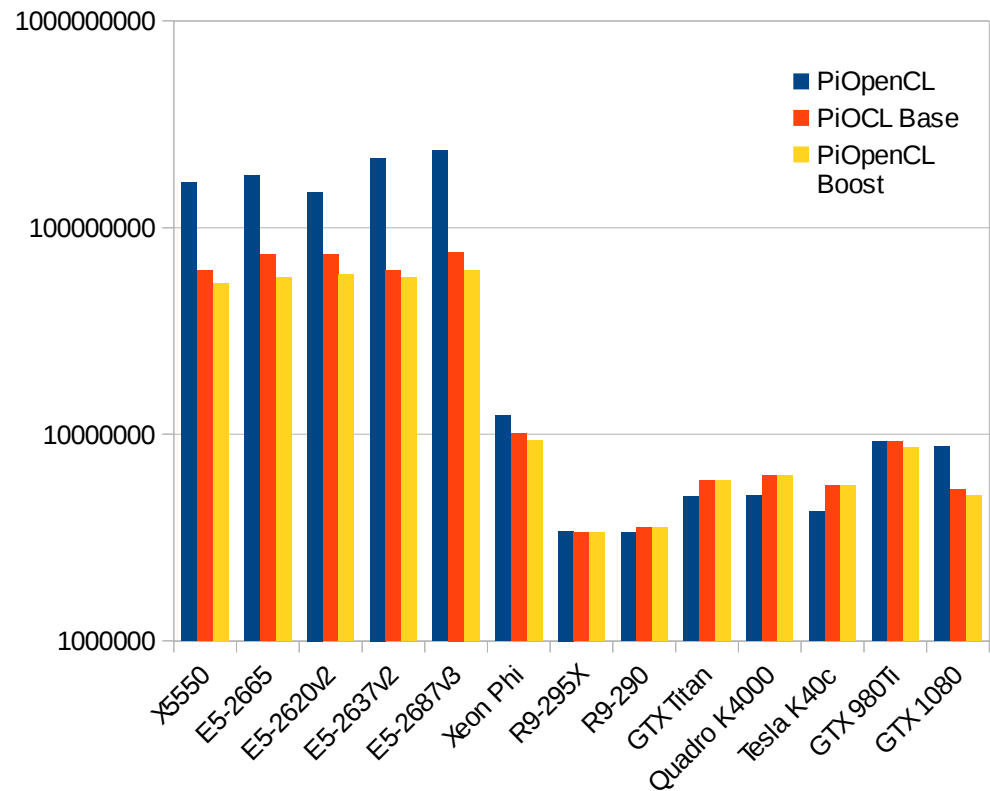
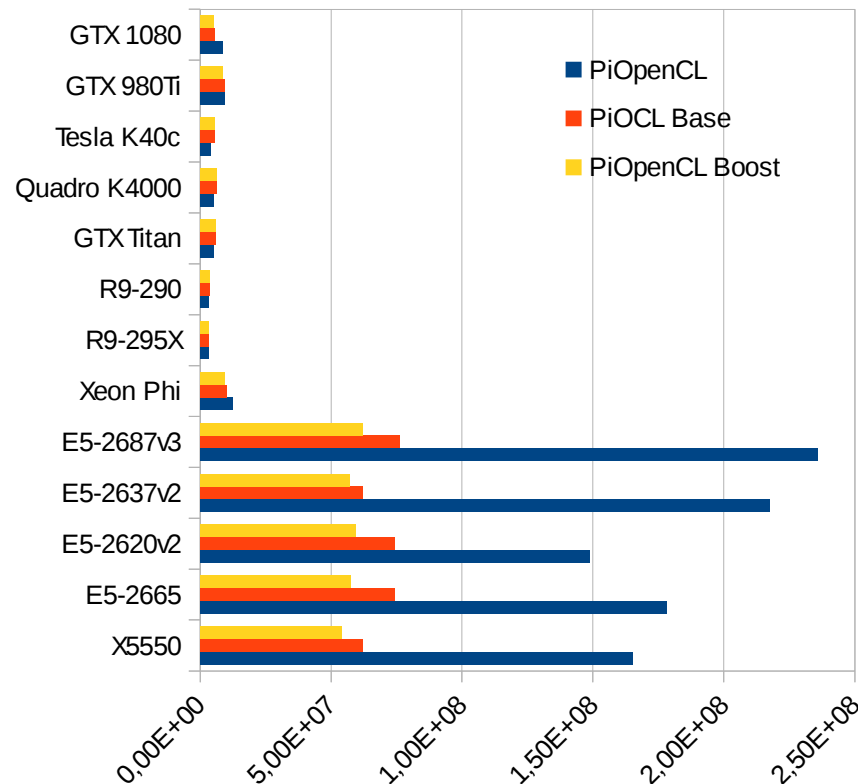
- Mais surtout des comportements différents !

Après le CPU et le GPU, Définition du QPU et EPU

- PR : « Parallel Rate » ou « Régime de Parallélisme »
- CPU : Central Processing Unit
- GPU : Graphical Processing Unit
- QPU : Quantum Processing Unit
 - Pour un « usage », lancement avec PR=1
 - Exemple : lancement pour Threads=1, Blocks=1 ou Work Items=1
- EPU : Equivalent Processing Unit
 - Pour un « usage », optimum de puissance pour un PR donné
 - Exemple : pour un R9-290x, optimum pour 8x le nombre de « shaders »

Avec 1 QPU, à bas « régime »... Le CPU enfonce le GPU !

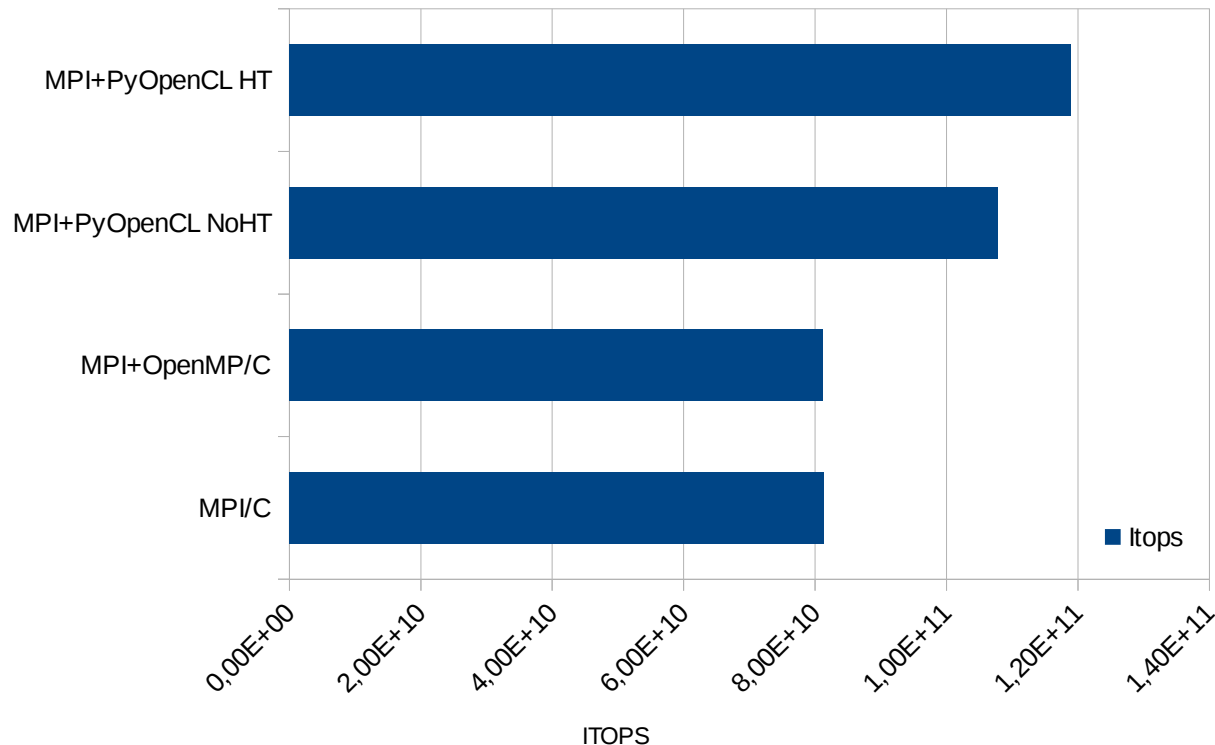
- De 20 à 50x plus lent !



- 1,5 ordre de grandeur...

Python est-il efficace ?

Une rapide comparaison...

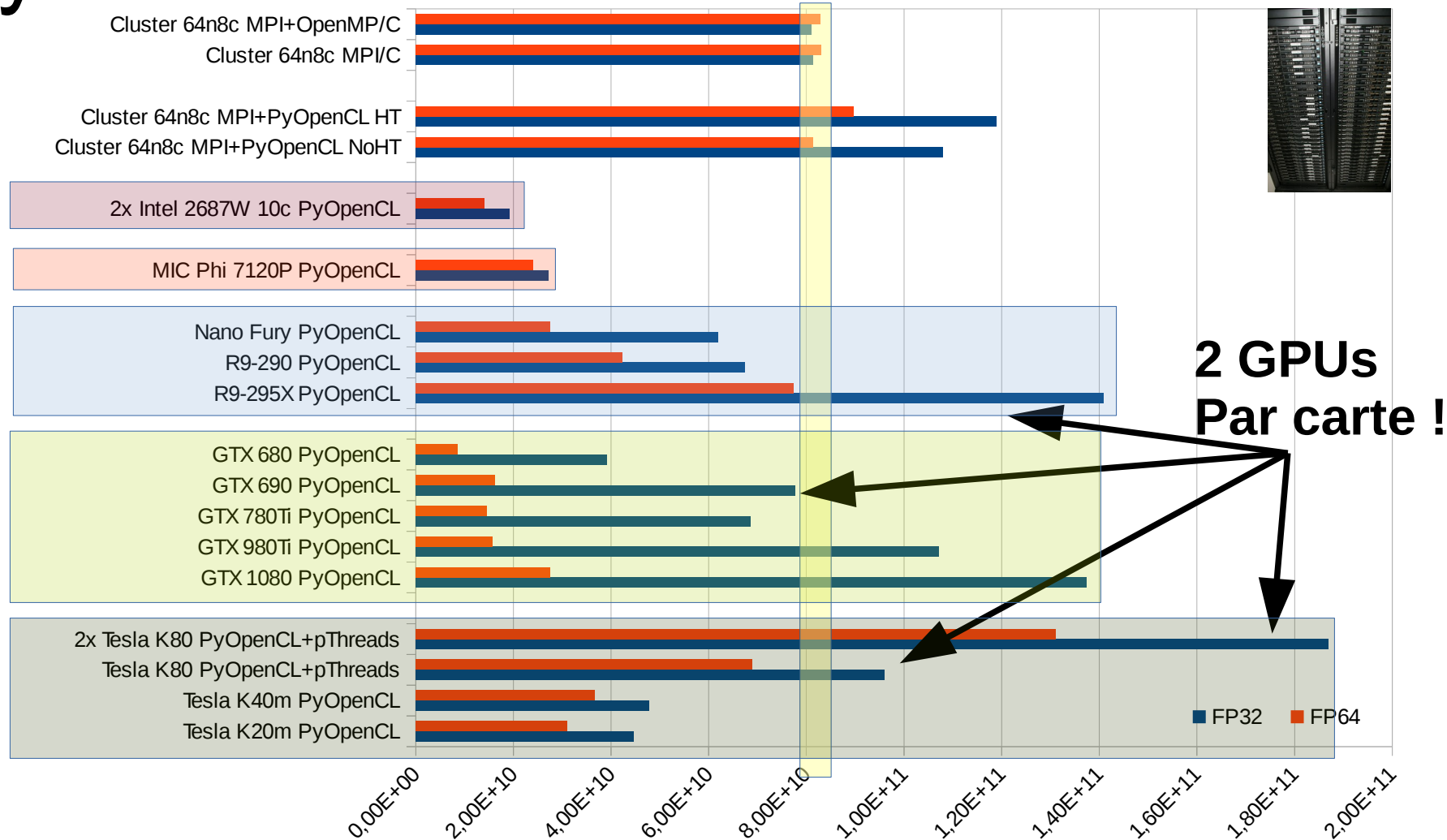


- 64 nœuds avec 512 cœurs
- 3 implémentations (2 hybrides)



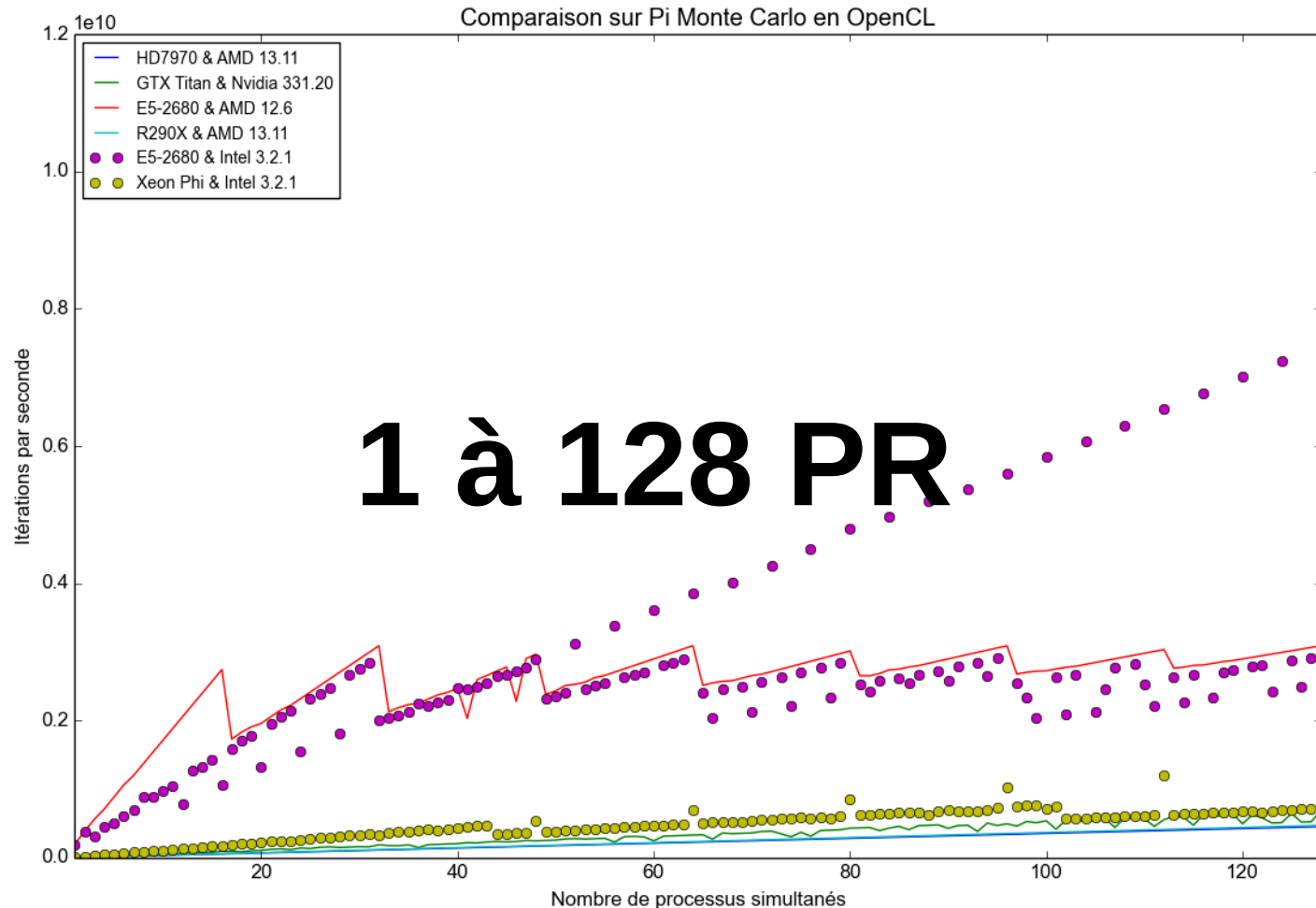
Pi Monte Carlo : le match...

Python vs C & CPU vs GPU vs Phi

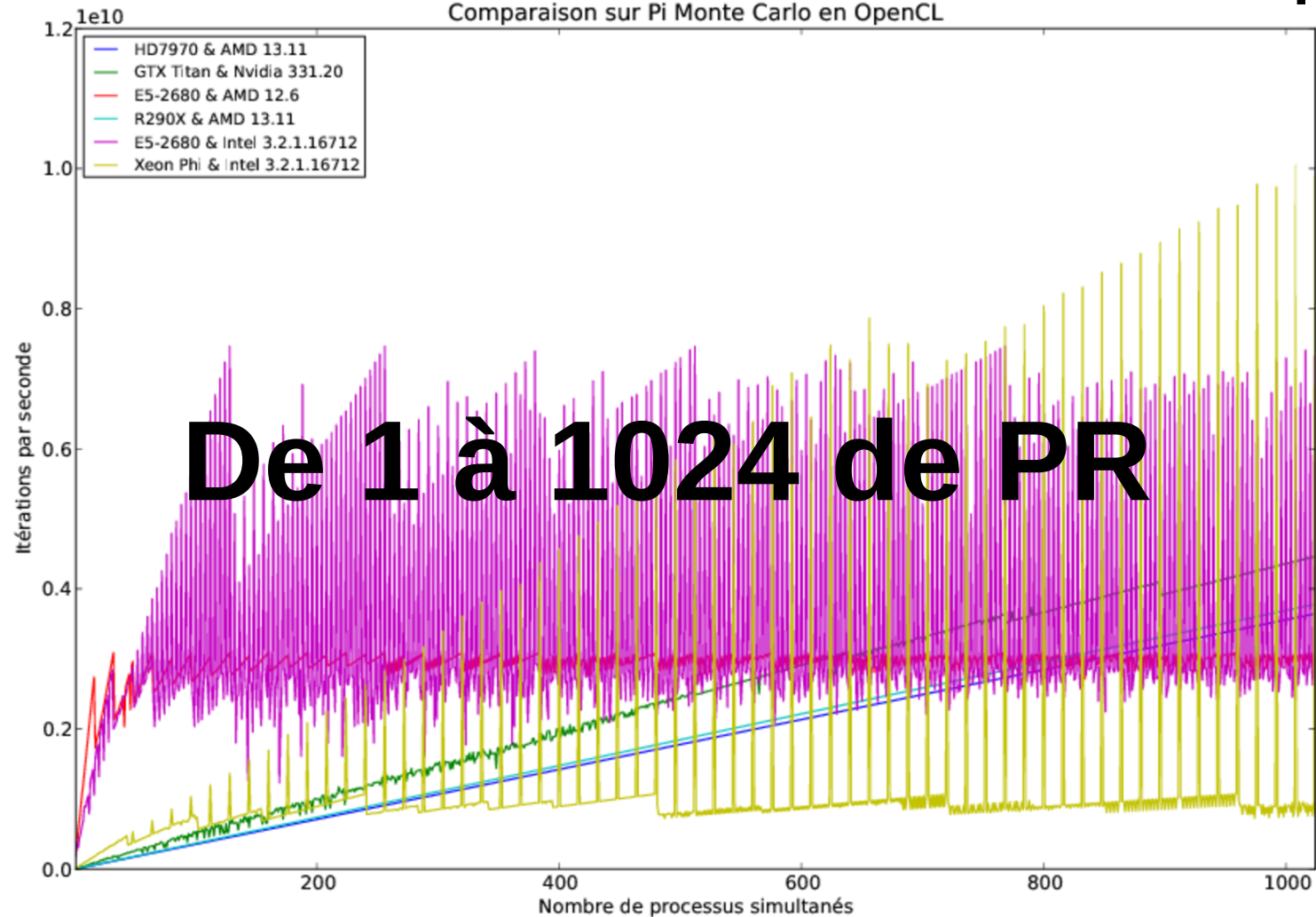


Petite comparaison entre *PU

Bas pararégime : le règne du CPU

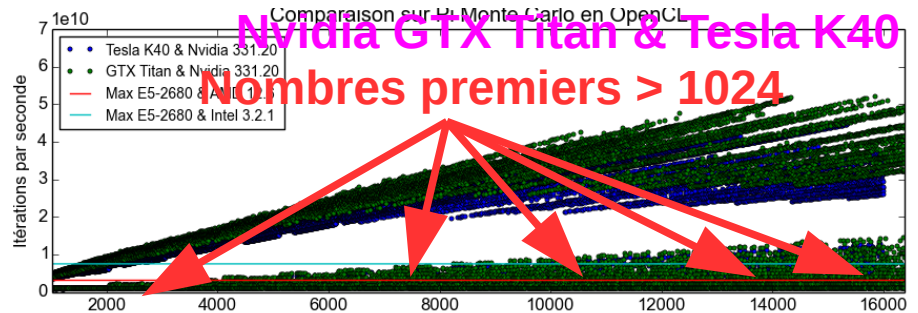
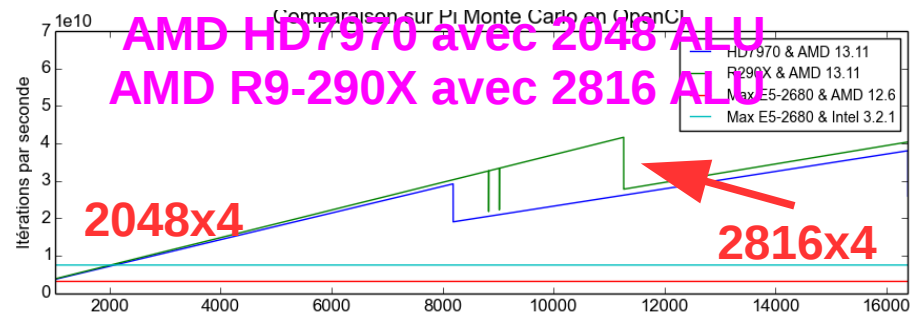
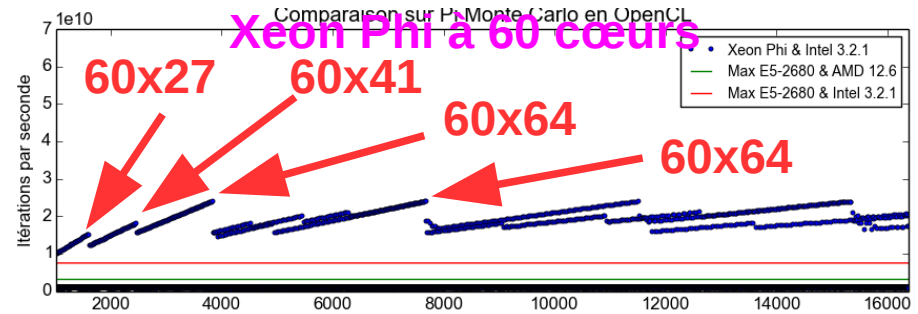
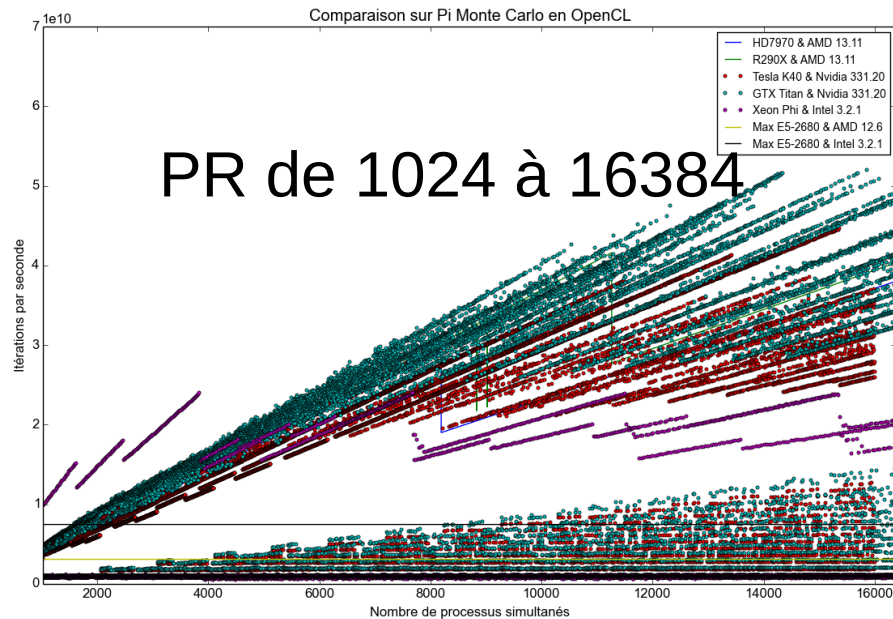


Petite comparaison entre *PU MIC Phi sort du bois, GPU embusqué



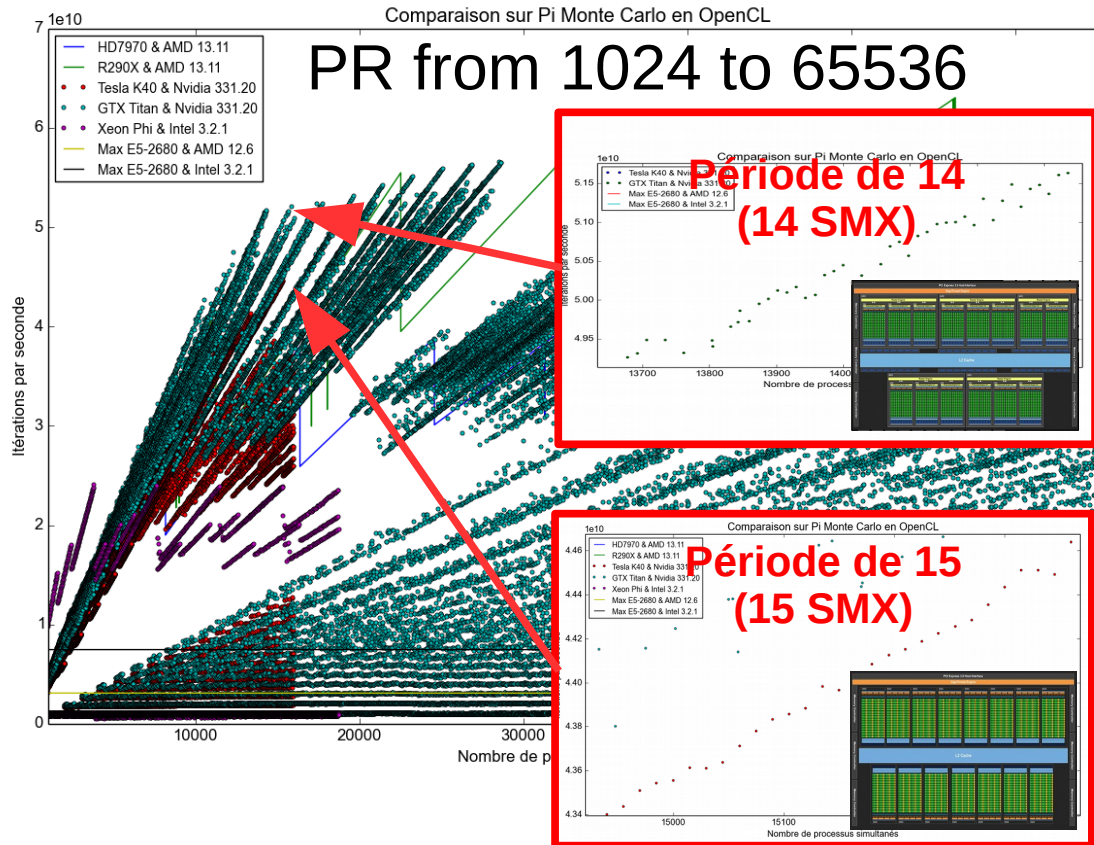
Exploration profonde de PR

ParaRégime de 1024 à 16384



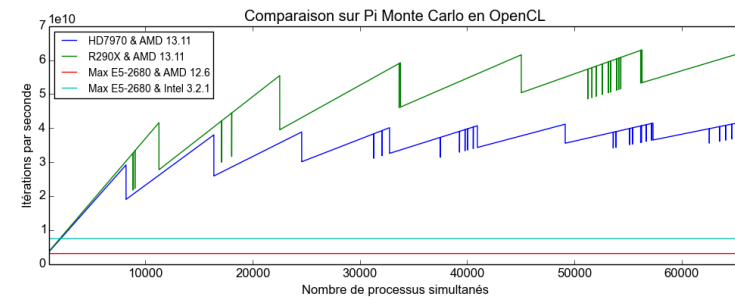
Exploration très profonde des PR

Régime de //isme de 1024 à 65536



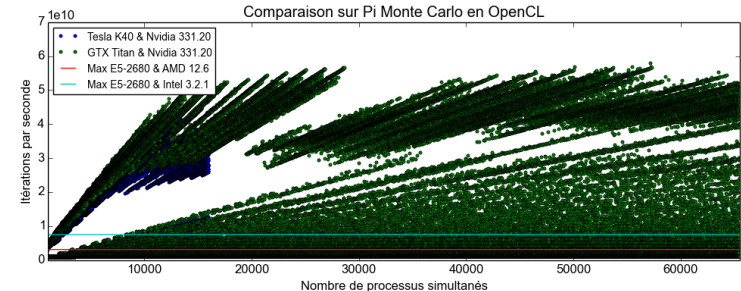
AMD HD7970 & R9-290

Longue Période : 4x nombre d'ALU

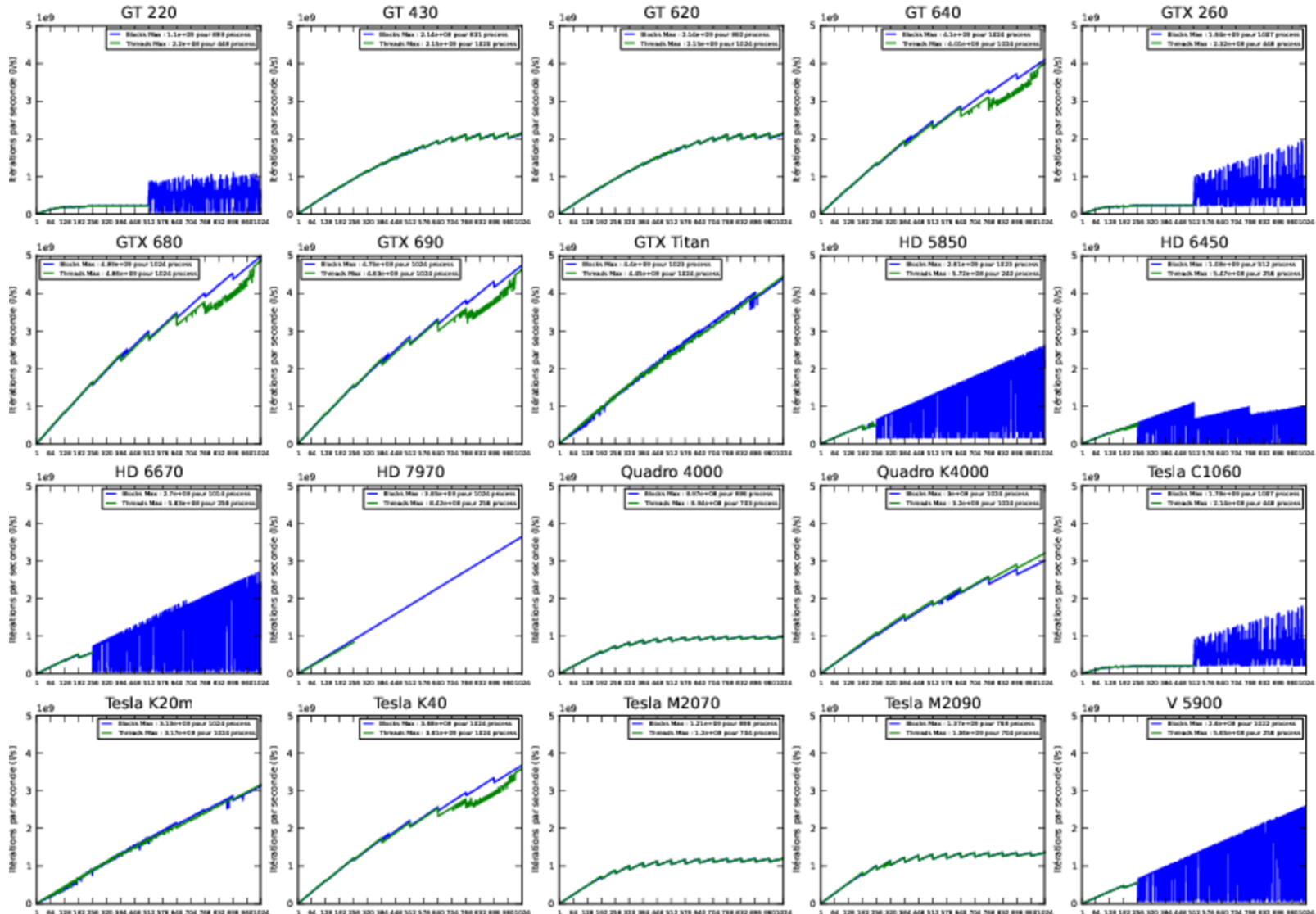


Nvidia GTX Titan & Tesla K40

Courte Période : nombre d'unités SMX

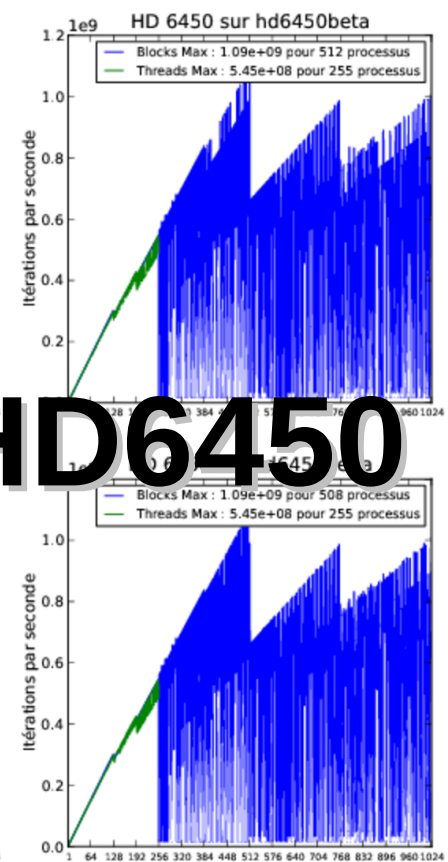
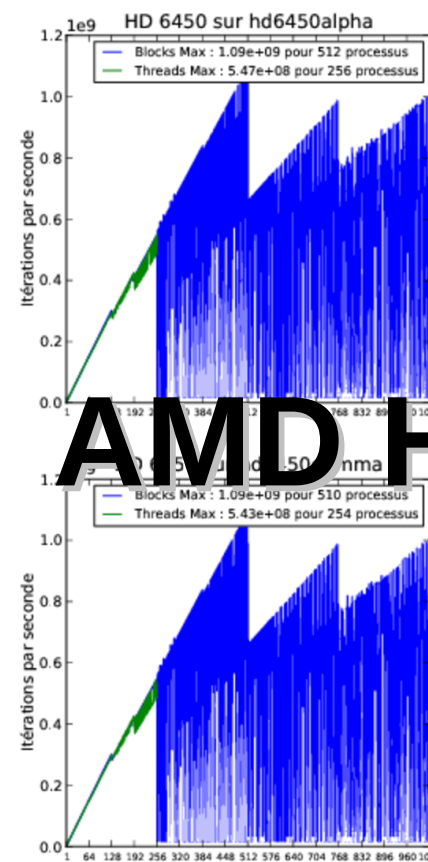
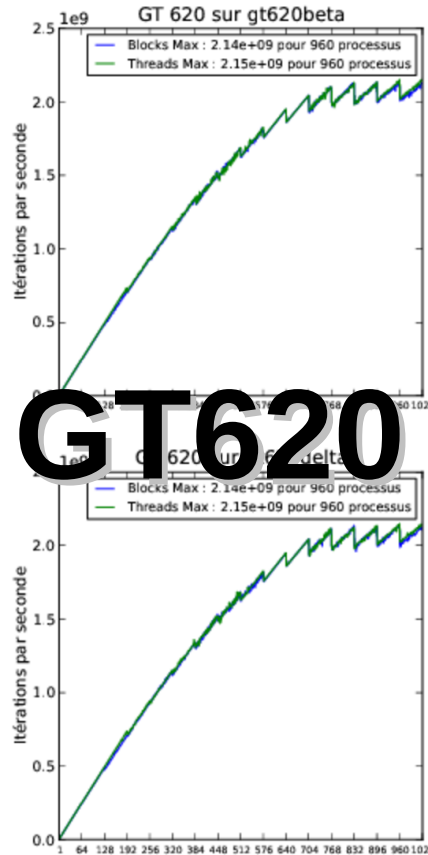
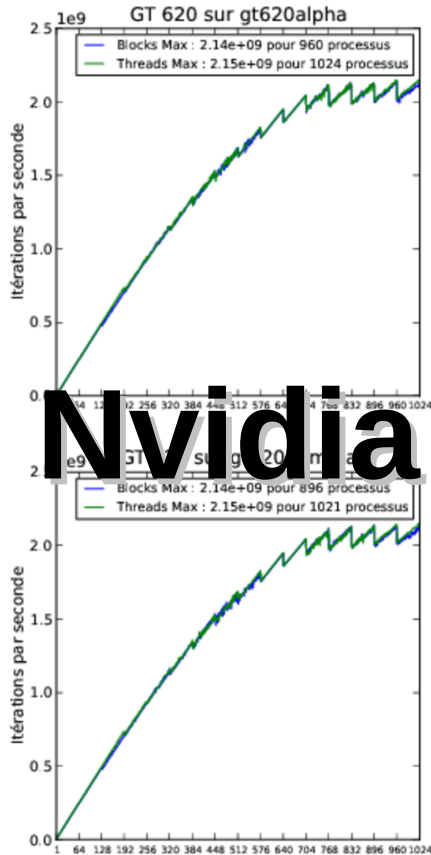


Attitudes de 20 (GP)GPU vs PR



Enveloppes de //isme reproductibles ?

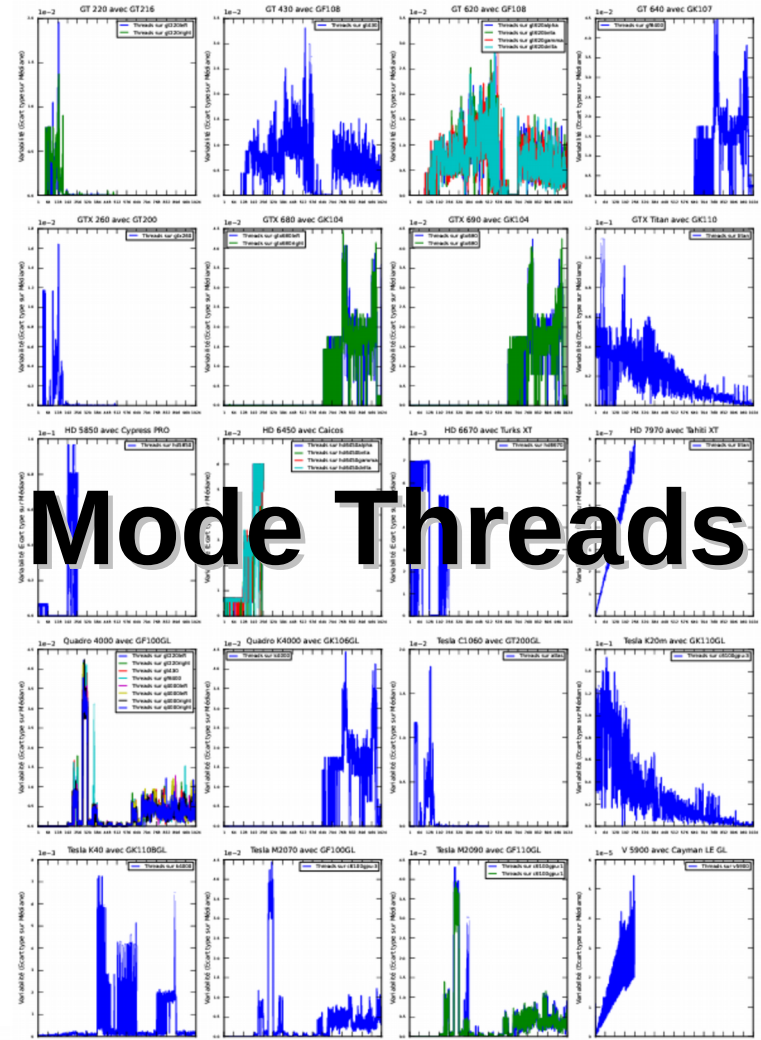
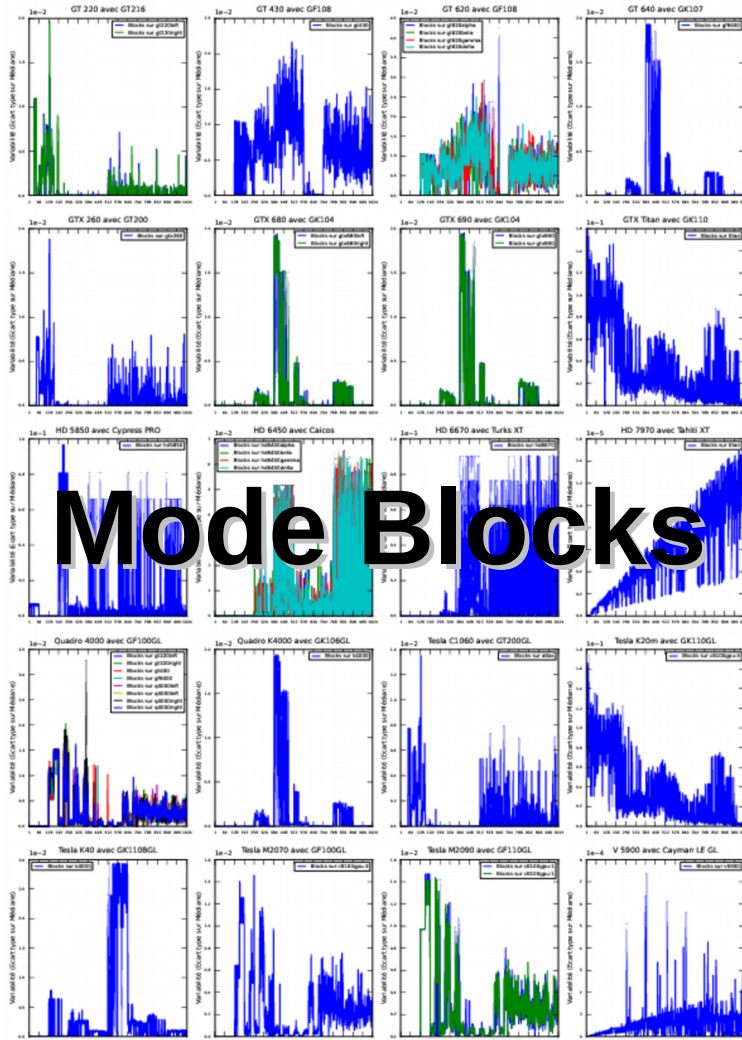
Pour deux cartes différentes...



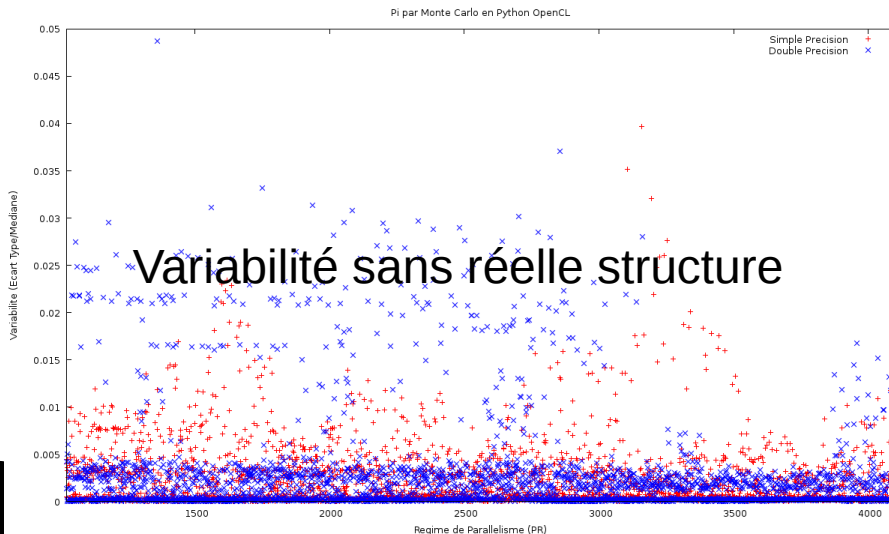
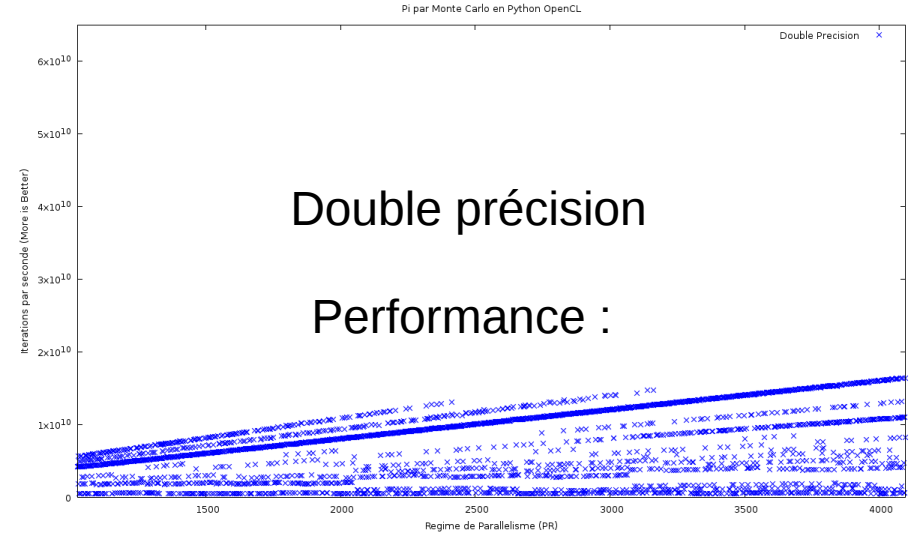
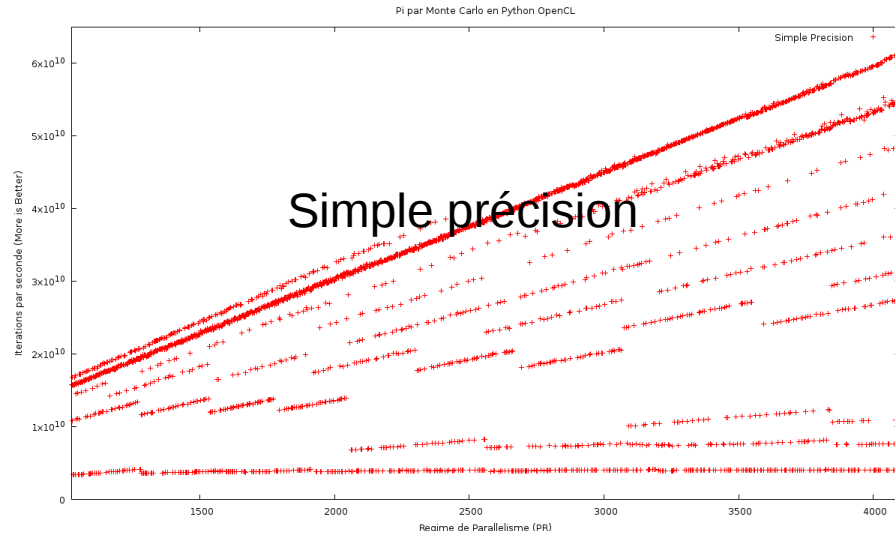
Nvidia GT620 AMD HD6450

Variabilité : un critère distinctif !

Quelle étrange propriété :-/ ...



Et la dernière Nvidia GTX1080, Toujours affectée de bizarreries ?



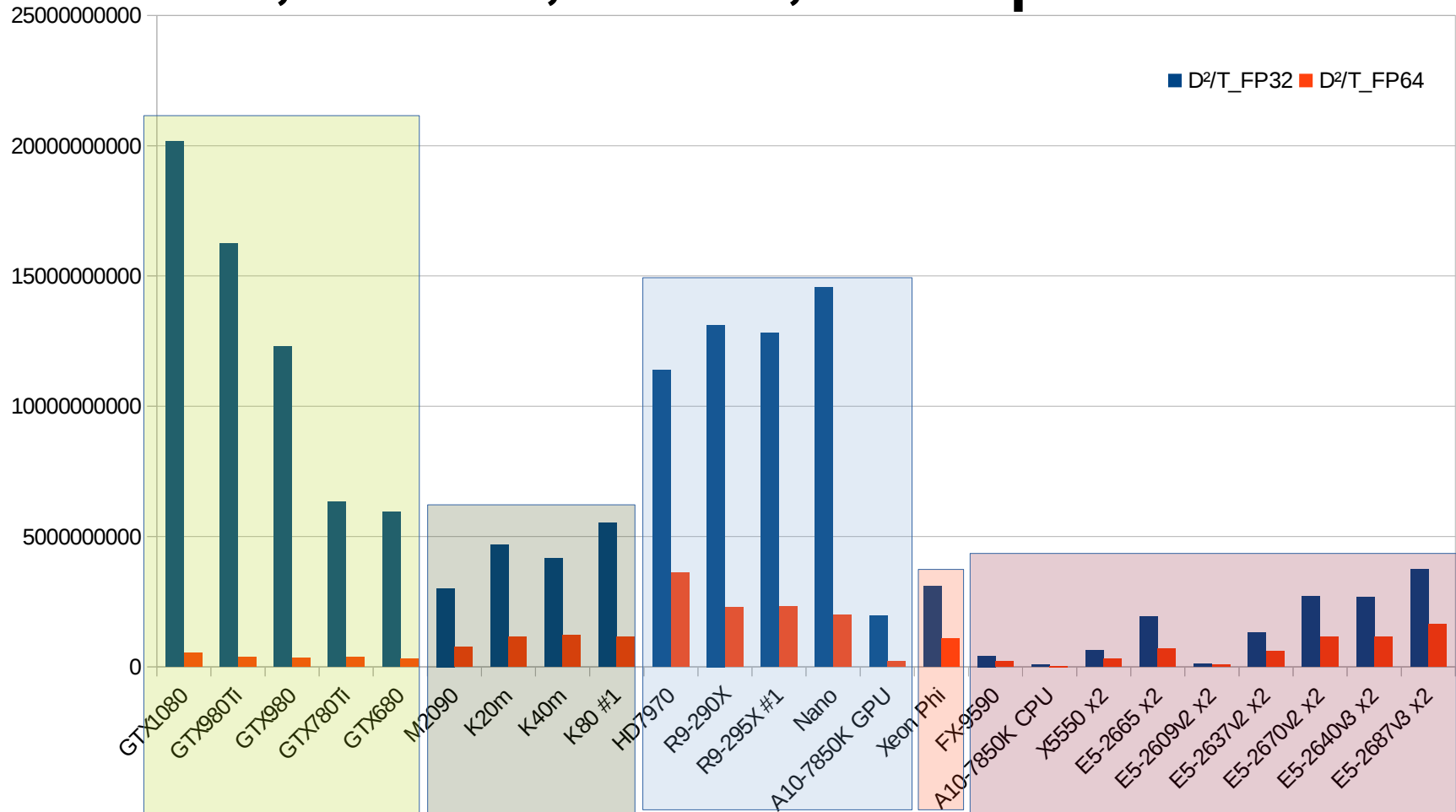
« Back to the Physics »

Un code N-corps newtonien...

- Passer sur un code « grain fin »
- Code N-corps très simplement intégré
 - 2ème loi de Newton, système autogravitant
- Méthodes d'intégration différentielles :
 - Euler Implicite, Euler Explicite, Heun, Runge Kutta
- Paramètres :
 - IPhysique : nombre de particules
 - Numérique : pas d'intégration, nombre de pas
 - Puis : influence FP32, FP64

Des performances très différentes

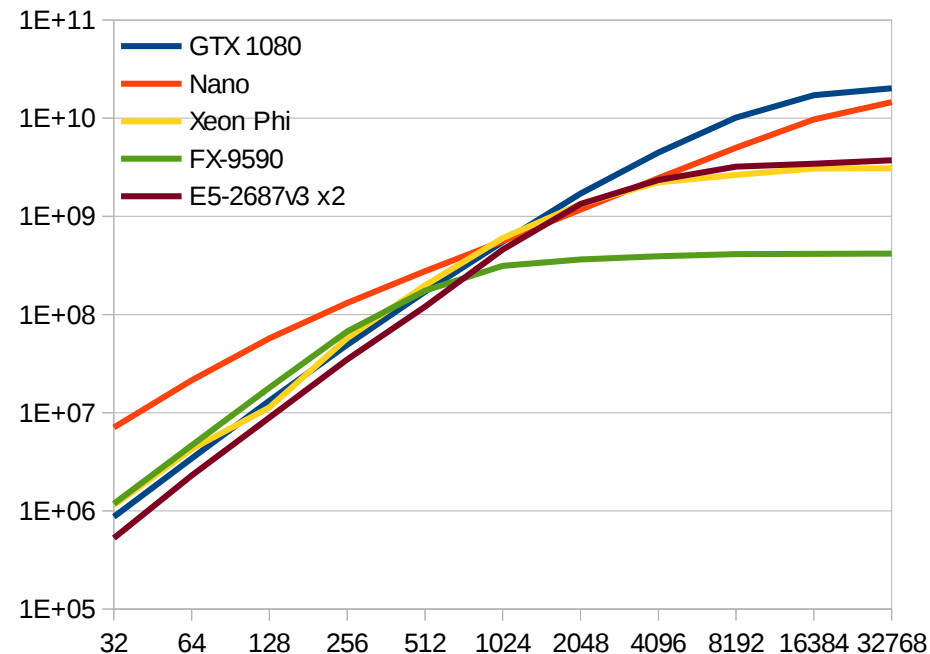
Nvidia, AMD, Intel, Simple/Double



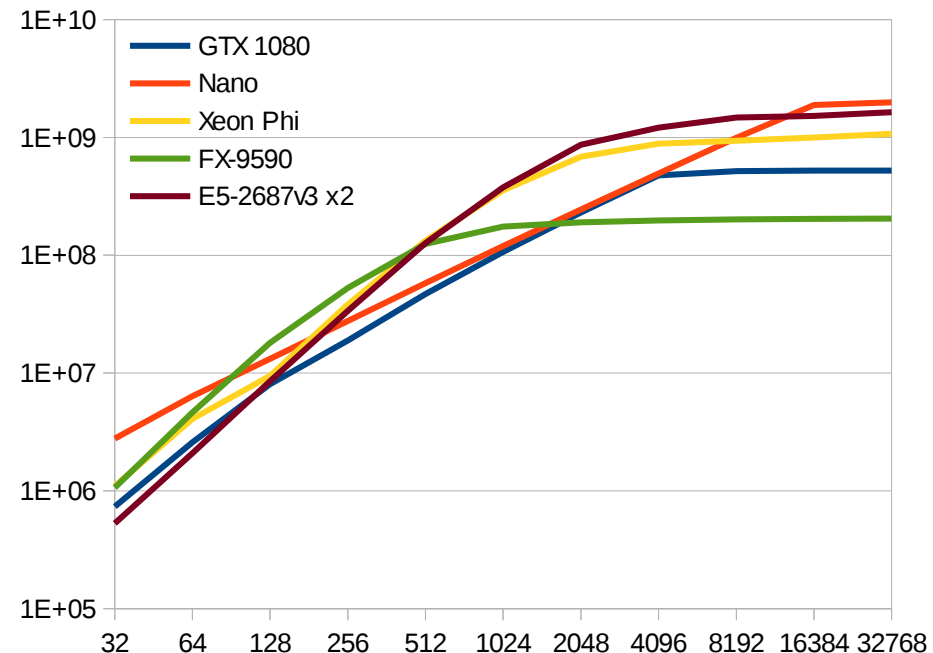
Pour les *PU les plus puissants

De la « charge » à la saturation...

Simple précision



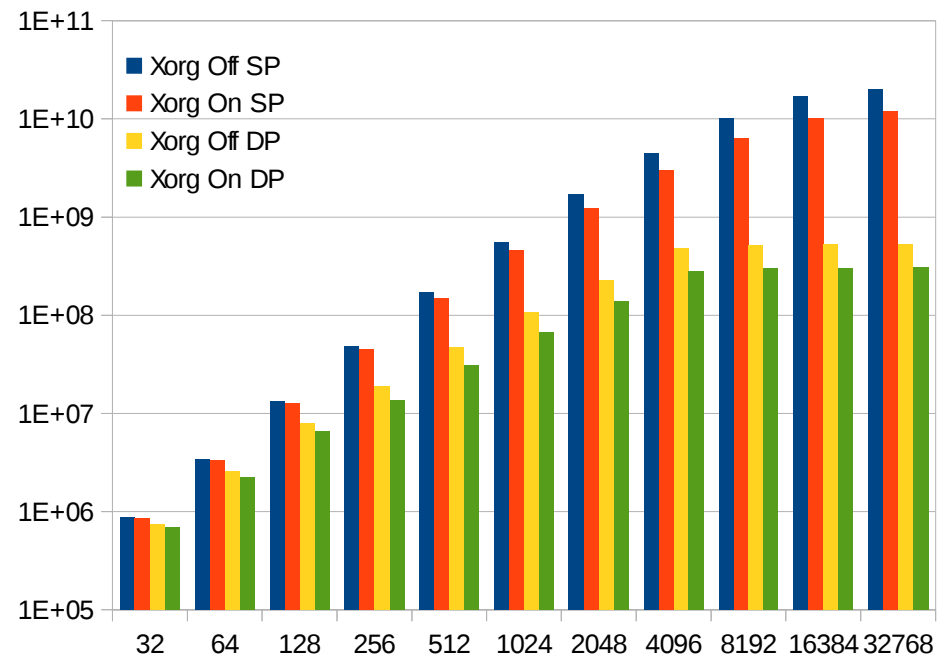
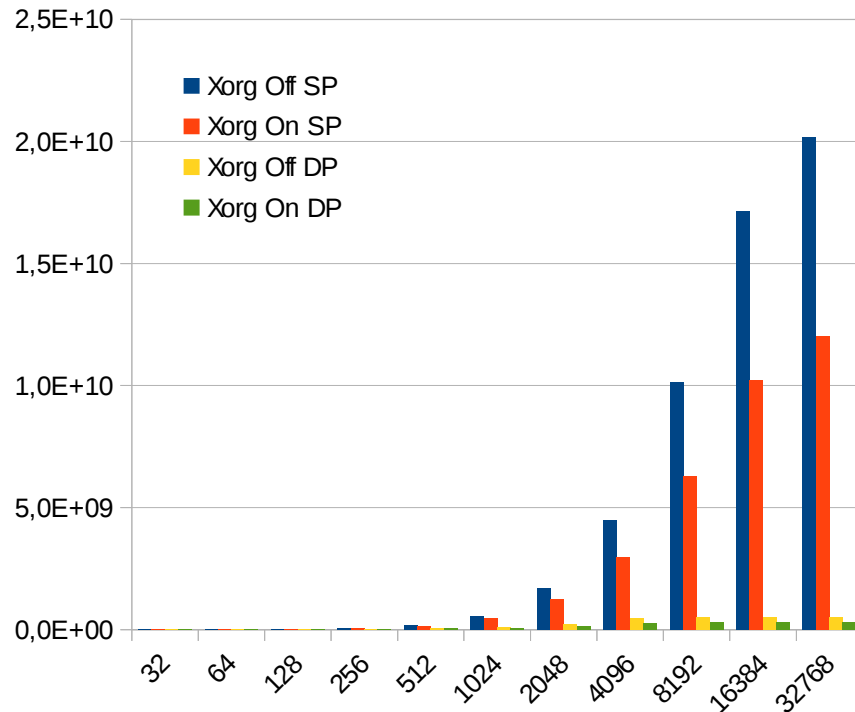
Double précision



- Résultats exprimés en Taille²/Durée

Mais si un GPU est si puissant...

Quid de sa fonction originelle ?



- En cas d'usage en GPGPU, attention au Xorg !
- Pour les AMD/ATI, pénible hors de l'usage root...

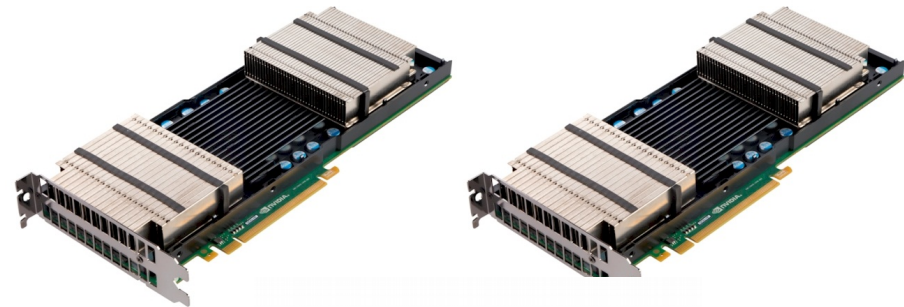
Comment « partager » un GPU ?

La virtualisation matérielle...

- Virtualisation : du partage de ressources...
 - Partage classique de CPU, mémoire, réseau (VT-x, VMX, SVM)
 - Partage de périphériques via VT-d : passage de PCI ou USB
- Avantages :
 - Rationalisation d'utilisation
 - Rapidité de déploiement
 - Isolation des ressources
- SIDUS pour le déploiement d'OS sans disque.

Cas d'usage : partage de 2 K80

- Sur l'hôte :
 - BIOS : Activation VT-x, VT-d
 - OS : intel_iommu=on
 - PCI-stub des unités PCI
- Sur les VM sous KVM :
 - Association d'1 unité PCI
 - SIDUS comme OS
 - Utilisation directe de la carte
- Disponibilité : 60 secondes !



Mais à quoi ça sert tout ça ?

Caractériser & éviter les regrets...

- Ce qu'il faut retenir :
 - Dans une machine, en 2016, la puissance « brute » est dans le GPU
 - Pour un régime de parallélisme bas, le CPU enfonce le GPU
 - Un GPU n'est supérieur au CPU que pour $PR > 1000$
 - Le GPU n'atteint son optimum QUE pour certains PR
 - Sans une caractérisation, des déceptions à prévoir...
 - OpenCL est un bon compromis comme langage *PU
 - Python reste l'approche la plus rapide de OpenCL
- Ce qu'il faut faire : **expérimenter !**

Il y a 10 ans, à l'ENS-Cachan...

Proposition de pistes...

- Convergence entre processeurs
 - Maintenant, les processeurs les plus récents sont sur les personnels
- Exploitation de la virtualisation dans l'apprentissage
 - Large exploitation pour les machines à la demande
- Utilisation de Python comme langage scientifique
 - Premier langage utilisé en science dans les universités au monde
- Utilisation des cartes graphiques en soutien
 - 4/10 premiers du Top 500 ont des accélérateurs
- Utilisation des algorithmes génétiques pour compilation
 - Le « machine learning » commence à envahir l'écosystème des codes...