



Conférence au CEA

De la GÉPUfication d'un vieux code à
l'émergence de bonnes pratiques de portage GPU

*“Un code pour les exploiter tous,
Un code pour les évaluer,
Un code pour les confronter tous...”*

Emmanuel Quémener

Du triptyque du projet... ... au triptyque des coûts

- Pour une gestion de projet simplifiée : les 3 questions...
 - **Où en sommes-nous** ? Etat des lieux : matériel, OS, logiciel, personnes
 - **Où allons-nous** ? Meilleure « performance » mais laquelle ?
 - **Comment y allons-nous** ? Développer, intégrer, adapter, maintenir...
- Face à la gestion des trois coûts :
 - **Coût d'entrée** : appropriation de l'environnement matériel & logiciel
 - **Coût d'exploitation** : MCO et évolution dans cet environnement
 - **Coût de sortie** : sortie ou migration de cet environnement

Tous un peu dans la même galère...

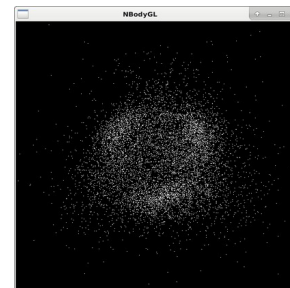
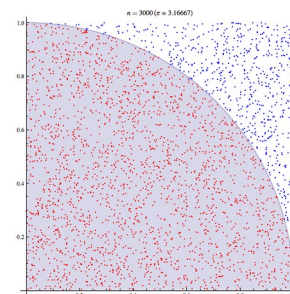
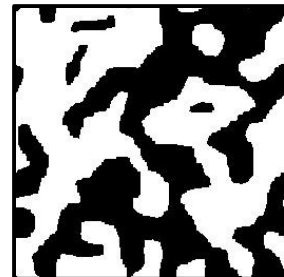
A l'origine, une notion essentielle en sciences : la « référence »

- Histoire personnelle : retour au HPC fin 2009
- Déferlante des (GP)GPU par Nvidia : GTX295 ou T1060
- Code de référence : LinPack du Top 500
 - Exercice personnel : portage du hpl de BLAS à CuBLAS
 - Finalement, plutôt inefficace face aux xBLAS : mais pourquoi ?
- Besoin de revenir aux « fondamentaux »
 - Écrire ses propres programmes exploitant les BLAS
 - Nombreuses implémentations : FBLAS, CBLAS, CuBLAS, clBLAS, gslBLAS...
 - Aborder les problèmes avec deux approches : intégrateur & développeur

Le souci de l'universalité...

L'émergence de « codes matrices »

- Pyphi 2011 : modèle d'Ising, transition de phase
 - comparaison C, MPI, OpenCL, CUDA...
- Fin 2012 : code Pi Monte Carlo
 - Gros grain, simple, parallélisable de manière quasi-continue...
 - Charge sur le RNG, nature des variables (INT, FP, 32&64)
 - Implémentations sur toutes les approches parallélisées
- Fin 2014 : code Nbody
 - Grain fin, physique, parallélisable de manière aussi continue
 - Charge sur les calculs, nature des variables (FP32&FP64)



Mais aucune exploitation massive de la mémoire...

40 ans entre simulation et mesure

De 2019 à 1979...

1979 : JP Luminet A&A

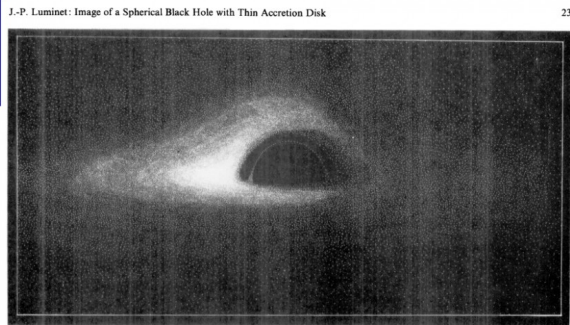


Fig. 11. Simulated photograph of a spherical black hole with thin accretion disk

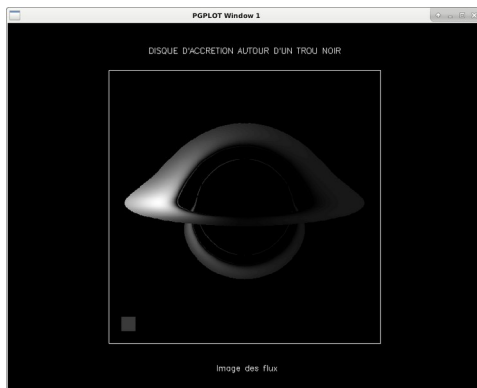
2014 : Film Interstellar



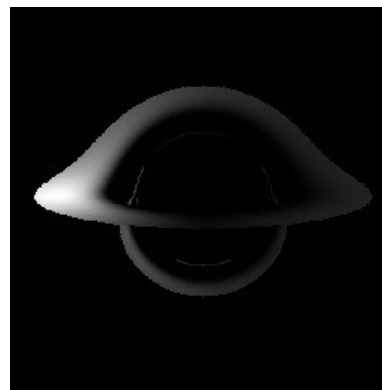
2019 : EHT, Messier 87



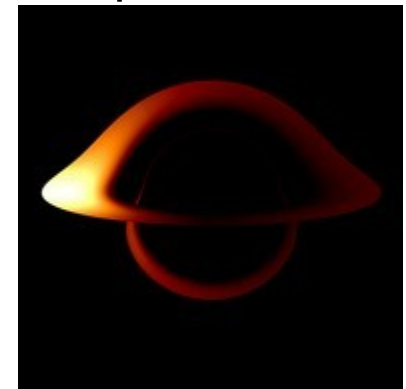
1994 : Code Fortran



1997 : Code C



2019 : OpenCL/CUDA



La physique de base

Tout dans un article de JP Luminet !

- Relativité générale d'Einstein
- Une métrique de Schwarzschild
- Réduction en équation polaire
- Dérivation de l'équation polaire
- Système du second ordre
- Modèle d'émission de disque
 - Raie monochromatique : cas d'école
 - Corps noir : modèle plus réaliste

Astron. Astrophys. 75, 228–235 (1979)

ASTRONOMY
AND
ASTROPHYSICS

Image of a Spherical Black Hole with Thin Accretion Disk

J.-P. Luminet

Groupe d'Astrophysique Relativiste, Observatoire de Paris, Section d'Astrophysique, F-92190-Meudon, France

Received July 13, 1978

Summary. Black hole accretion disks are currently a topic of widespread interest in astrophysics and are supposed to play an important role in a number of high-energy situations. The present paper contains an investigation of the optical appearance of a spherical black hole surrounded by thin accretion disk. Isoradial curves corresponding to photons emitted at constant radius from the hole as seen by a distant observer in arbitrary direction have been plotted, as well as spectral shifts arising from gravitational and Doppler shifts. By the results of Page and Thorne (1974) the relative intrinsic intensity of radiation emitted by the disk at a given radius is a known function of the radius only, so that it is possible to calculate the exact distribution of observed bolometric flux. Direct and secondary images are plotted and the strong asymmetry in the flux distribution due to the rotation of the disk is exhibited. Finally a simulated photograph is constructed, valid for black holes of any mass accreting matter at any moderate rate.

Key words: black holes – accretion disks – geometrical optics

1. Introduction

The aim of the present paper is to provide a reply to the question that many people ask themselves about the optical appearance of a black hole.

In order to be visible a black hole has of course to be illuminated, like any ordinary body. One of the simplest possibilities would be for the black hole to be illuminated by a distant localized source which in practise might be a companion star in a loosely bound binary system. A more interesting and observationally important possibility is that in which the light source is provided by an emitting accretion disk around the black hole, such as may occur in a tight binary system with overflow from the primary, and perhaps also on a much larger scale in a dense galactic nucleus. The general problem of the optical appearance of black holes is related to the analysis of trajectories in the gravitational field of black holes. For a spherical, static, electrical field-free black hole (whose external space-time geometry is described by the Schwarzschild metric) this problem is already well known (Hagihara, 1931; Darwin, 1959; for a summary, see Misner et al., 1973 [MTW]). In Sect. 2 we give only a brief outline of it with basic equations, trying to point out the major features which will appear later. All our calculations are done in the geometrical optics approximation (for a study of wave-aspects, see Sanchez, 1977). In Sect. 3 we calculate the apparent shape of circular rings orbiting a non-rotating black hole and the results are depicted in Figs. 5–6. In Sect. 4 we recall the standard analysis by Novikov and Thorne

(1973) of the problem of energy release by a thin accretion disk in a general astrophysical context, focusing attention more particularly on the analytic solution for the surface distribution of energy release that was derived by Page and Thorne (1974) in the limiting case of a sufficiently low accretion rate. In terms of this idealized (but in appropriate circumstances, realistic) model, we calculate the distribution of bolometric flux as seen by distant observers at various angles above the plane of the disk (Figs. 9–11).

2. Image of a Bare Black Hole

Before analyzing the general problem of a spherical black hole surrounded by an emitting accretion disk, it is instructive to investigate a more simple case in which all the dynamics are already contained, namely the problem of the return of light from a bare black hole illuminated by a light beam projected by a distant source. It is conceptually interesting to calculate the precise apparent pattern of the reflected light, since some of the main characteristic features of the general geometrical optics problem are illustrated thereby.

The Schwarzschild metric for a static pure vacuum black hole may be written as:

$$ds^2 = -\left(1 - \frac{2M}{r}\right) dt^2 + \left(1 - \frac{2M}{r}\right)^{-1} dr^2 + r^2 (d\theta^2 + \sin^2 \theta d\phi^2) \quad (1)$$

where r , θ , and ϕ are spherical coordinates and the unit system is chosen such that $G=c=1$. M is the relativistic mass of the hole (which has the dimensions of length). In this standard coordinate system the horizon forming the surface of the hole is located at the Schwarzschild radius $r_s = 2M$.

One can take advantage of the spherical symmetry to choose the "equatorial" plane $\theta = \pi/2$ so as to contain any particular photon trajectory under consideration. The trajectories will then satisfy the differential equation:

$$\frac{1}{r^2} \left(\frac{dr}{d\phi} \right)^2 + \frac{1}{r^2} \left(1 - \frac{2M}{r} \right) = 1/b^2. \quad (2)$$

The second term in the left member can be interpreted as an effective potential $V(r)$, in analogy with the non-relativistic mechanics. The motion does not depend on the photon energy E and on its angular momentum L separately, but only on the ratio $L/E = b$, which is the impact parameter at infinity.

Let the observer be in a direction fixed by the polar angle ϕ_0 in the Schwarzschild metric, at a radius $r_0 \gg M$. The rays emitted by a distant source of light and deflected by the black hole intersect the observer's detector (for example a photographic plate) at a

De l'article au rapport

- Métrique de Schwarzschild :

$$ds^2 = - \left(1 - \frac{2M}{r}\right) dt^2 + \left(1 - \frac{2M}{r}\right)^{-1} dr^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2)$$

- Equation polaire :

$$\left(\frac{1}{r^2} \left(\frac{dr}{d\phi}\right)\right)^2 + \frac{1}{r^2} \left(1 - \frac{2M}{r}\right) = \left(\frac{\pi_t}{\pi_\phi}\right)^2 = \frac{1}{b^2}$$

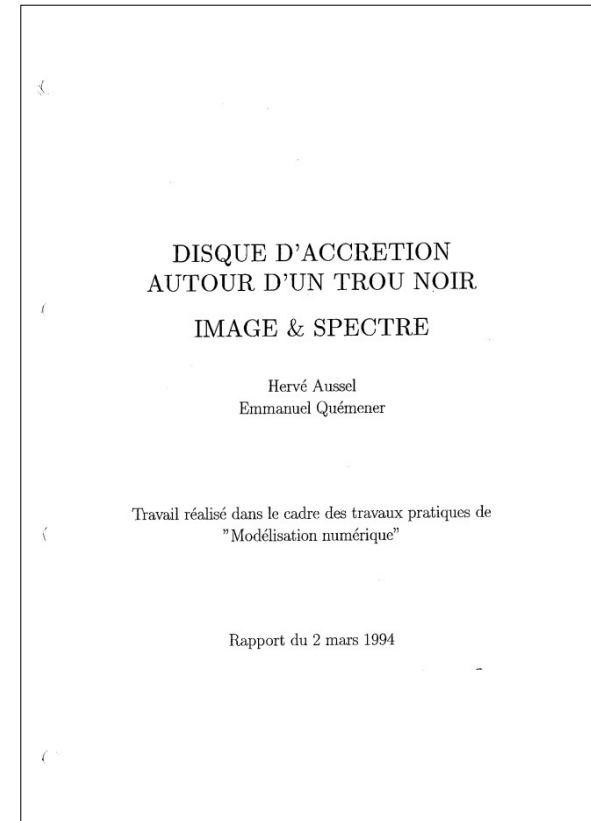
- Changement de coordonnées : $u=1/r$

$$\left(\frac{du}{d\phi}\right)^2 + u^2 \left(1 - \frac{2Mu}{b}\right) = 1$$

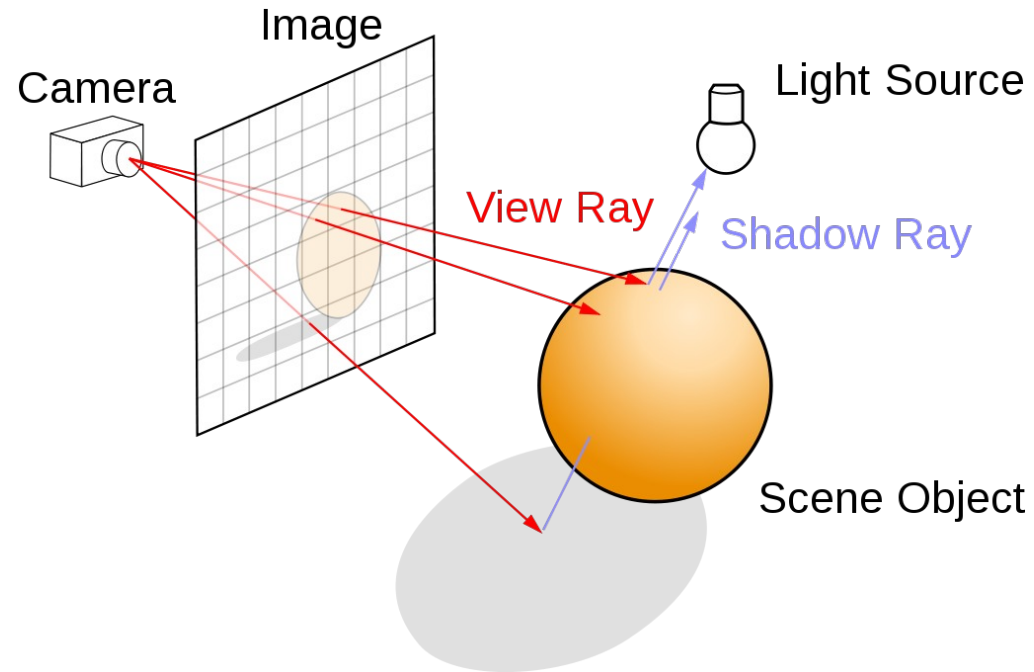
- Dérivation de l'équation polaire :

$$\frac{d^2u}{d\phi^2} + u \left(1 - \frac{3Mu}{b}\right) = 0$$

- Système d'équations à résoudre : $v = \frac{du}{d\phi}$ et $\frac{dv}{d\phi} = 3\frac{m}{b}u^2 - u$



Le « lancer de rayons » : Remonte le temps, forme une image



Source Wikipedia : Mental Ray CC BY-SA 3.0, Henrik CC BY-SA 4.0

Echarpe de plasma autour du Trou Noir

Pas « sans » mais « avec » dessus-dessous...

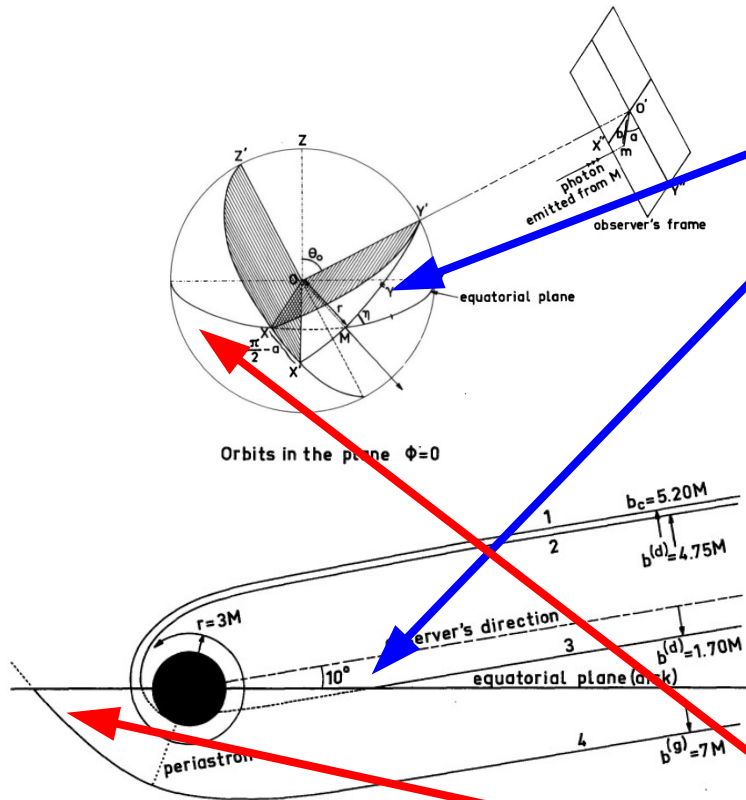
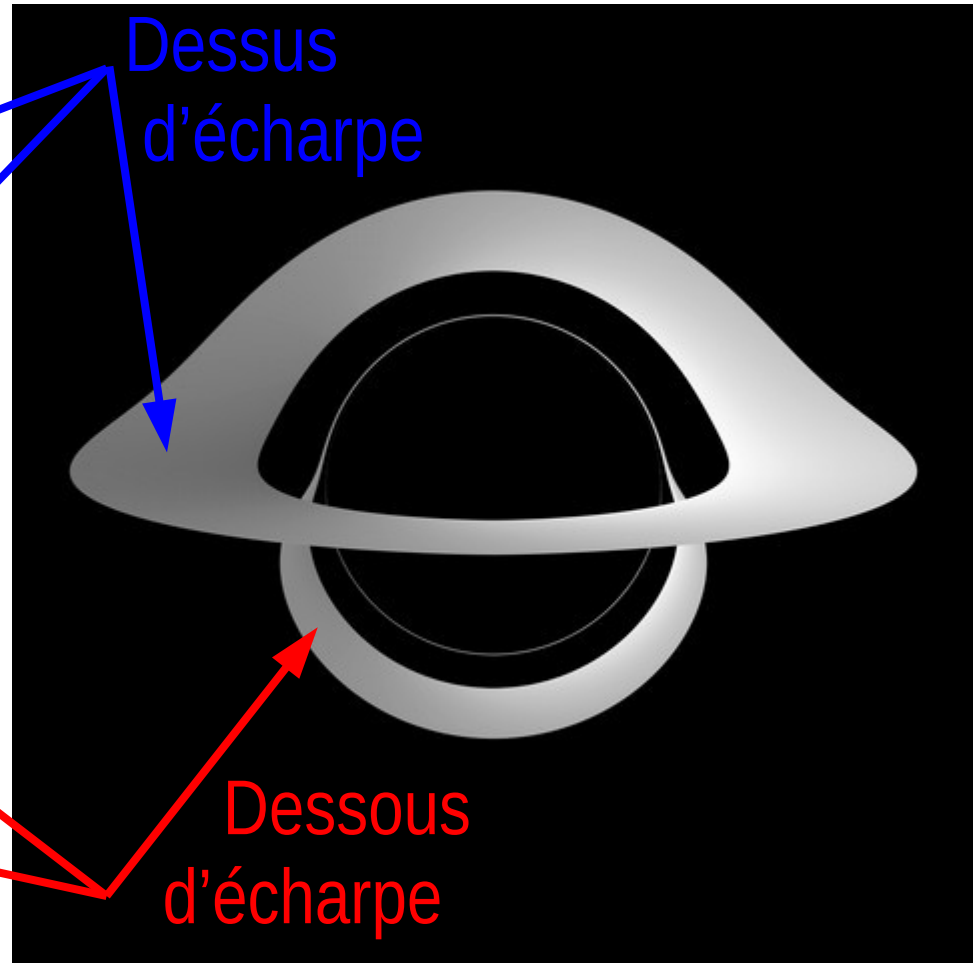
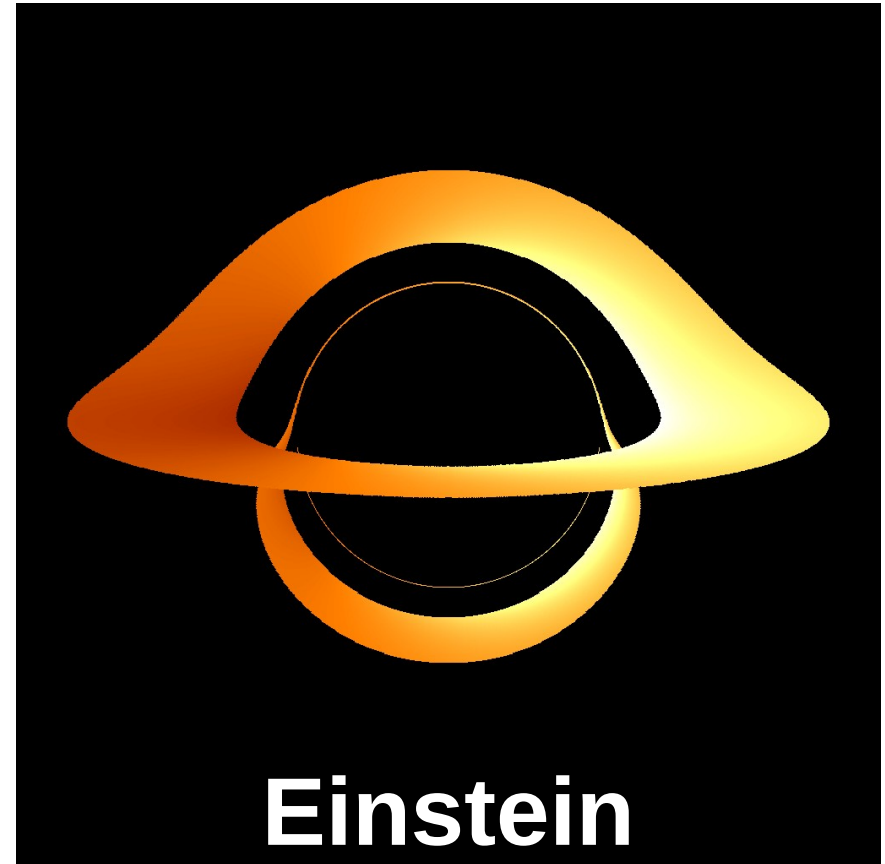
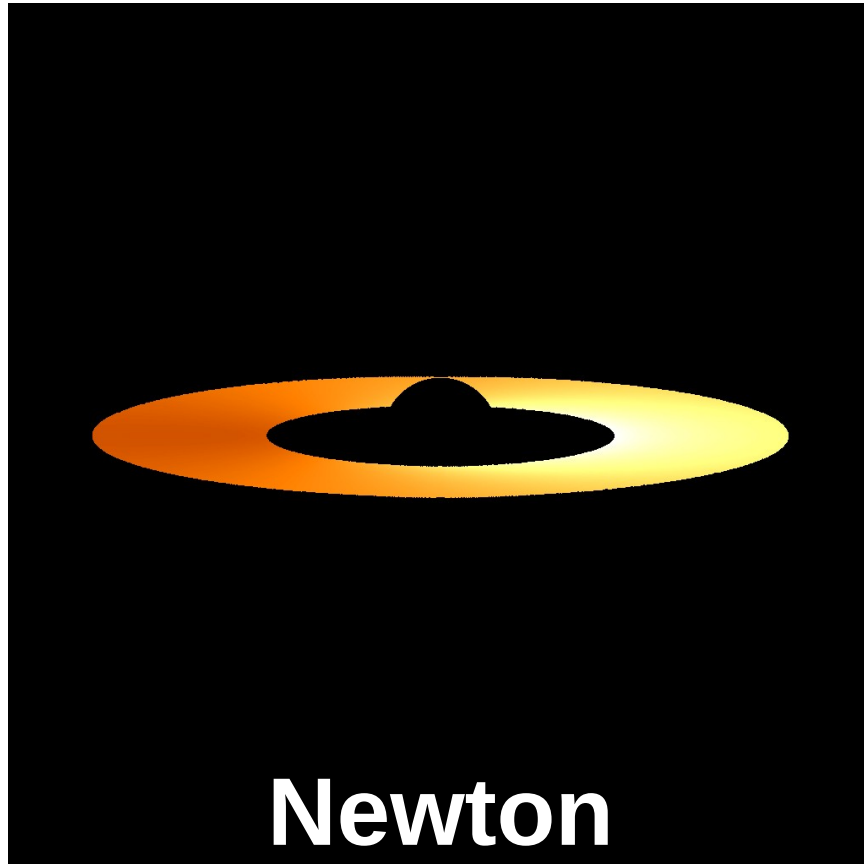


Fig. 4. Illustrative orbits in the plane $\{\phi=0\}$. Trajectory 1 has the critical impact parameter and circles infinitely around the black hole; trajectories 2 and 3 give direct images, trajectory 4 gives a secondary image

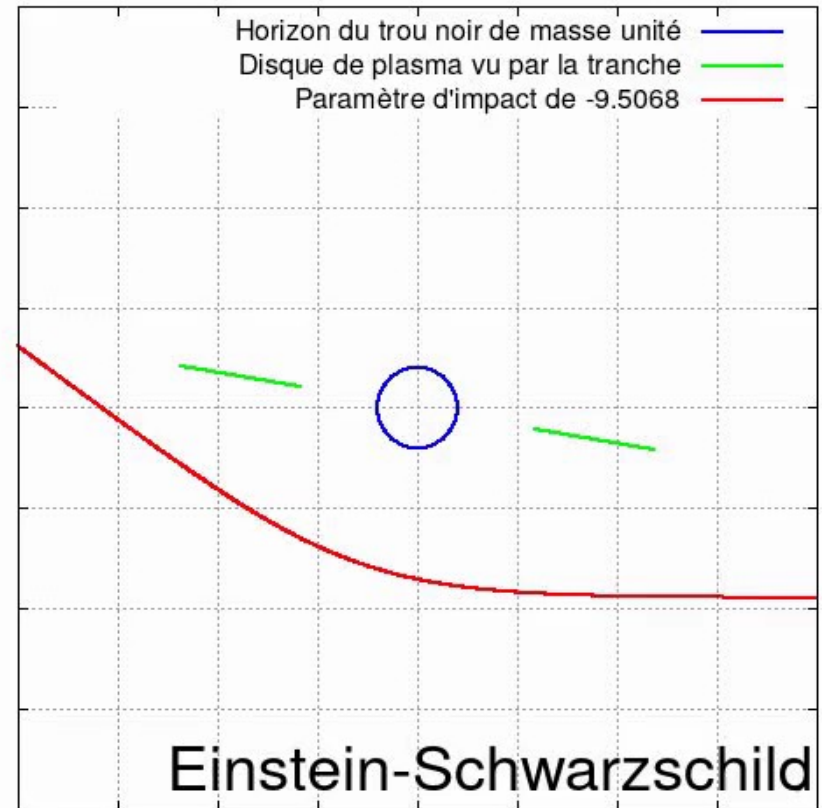
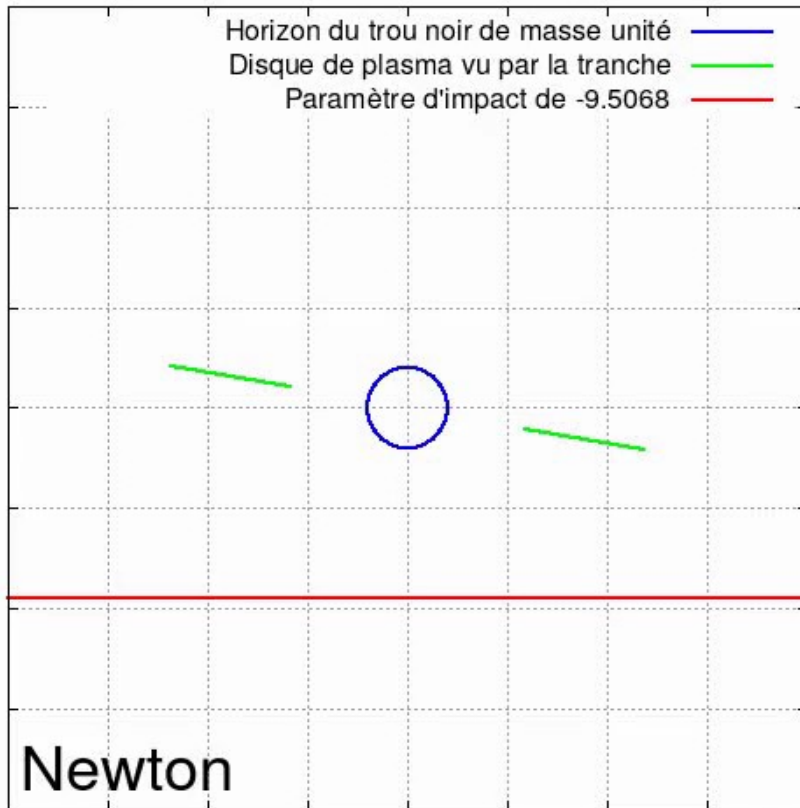


Calculer les trajectoires inverses

Entre Newton et Einstein

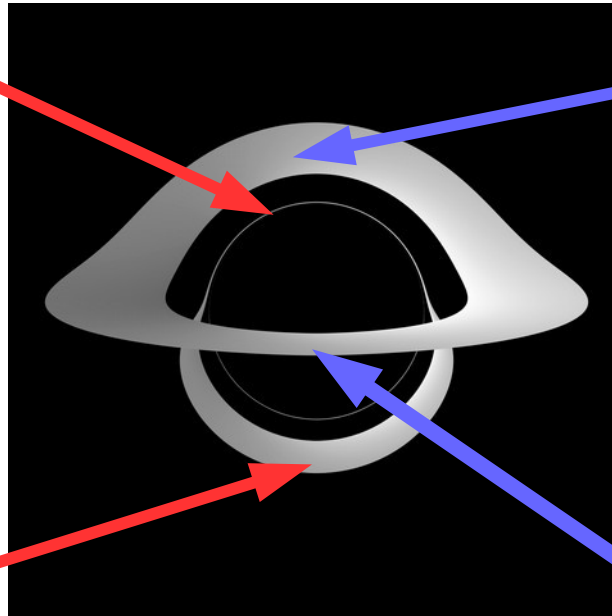
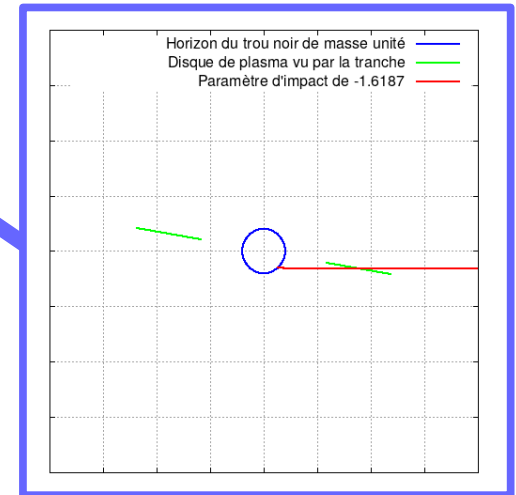
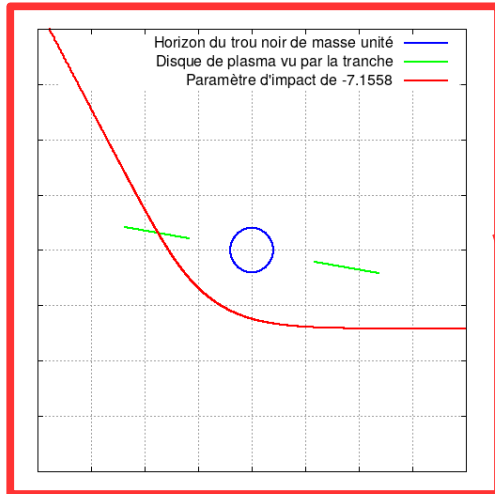
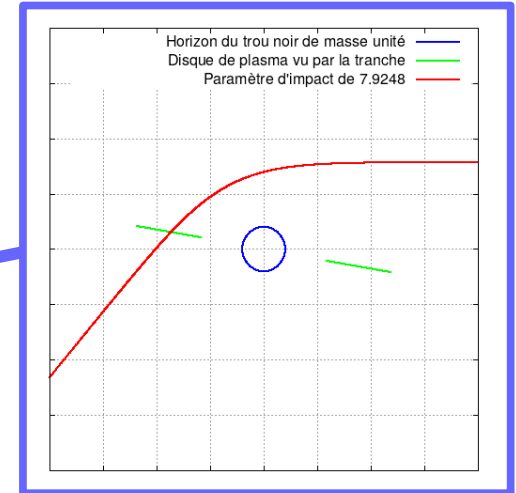
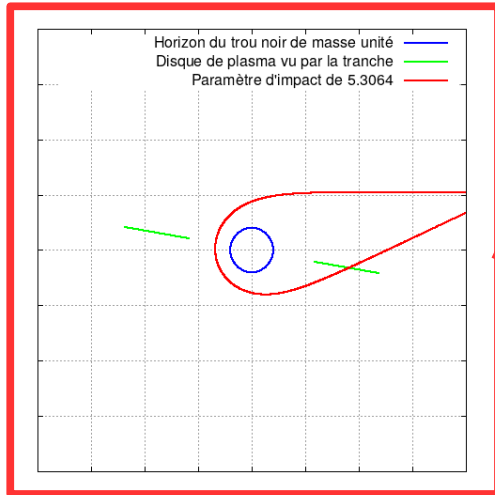


Et ça donne quoi pour tous les paramètres d'impacts ?

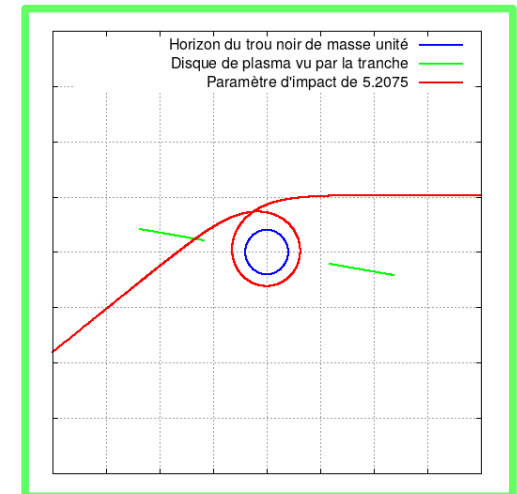
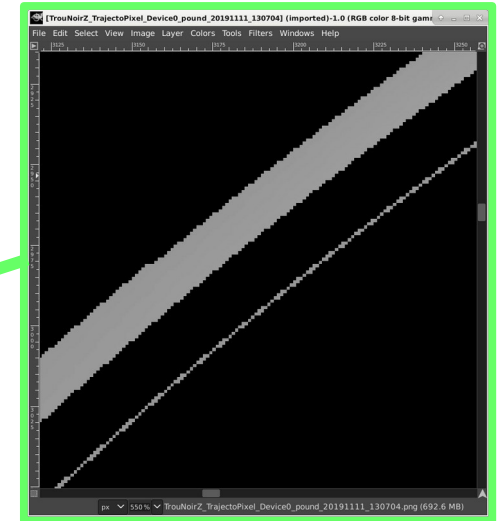
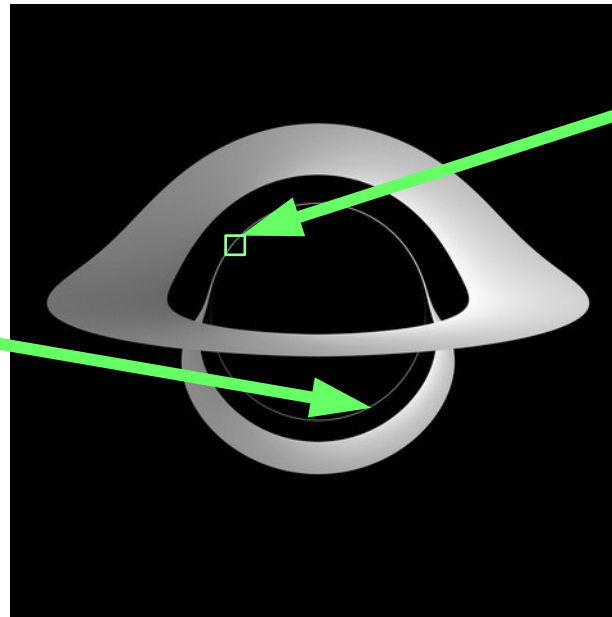
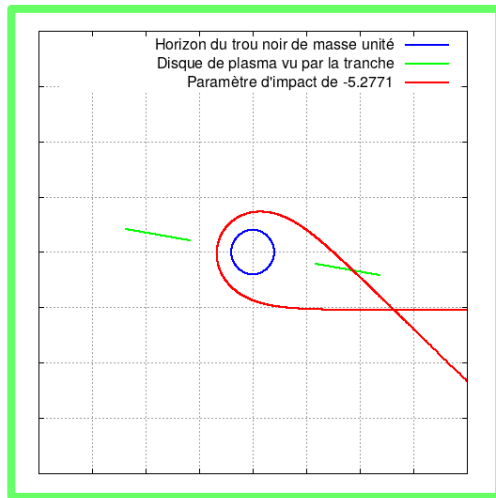


Copyright James MYLQ 2020

Quelques cas particuliers : le dessus, le dessous...



Quelques cas particuliers : le redessus, dessus et dessous

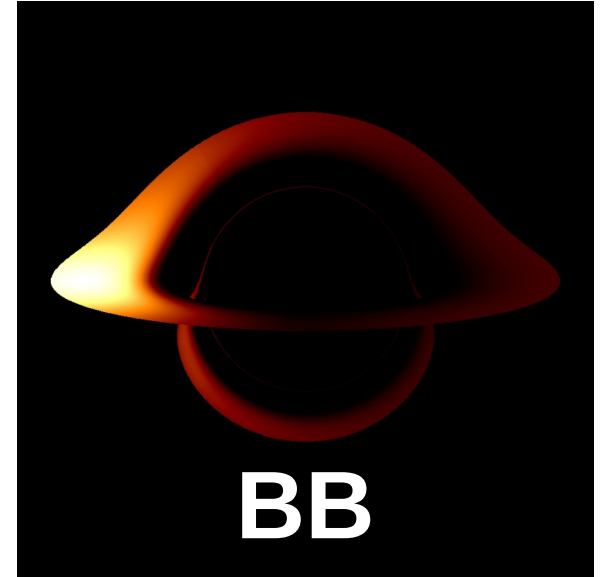


A chaque impact, deux physiques

Monochrome & Corps noir

« Puissance 4 » sur z

Spectre de planck



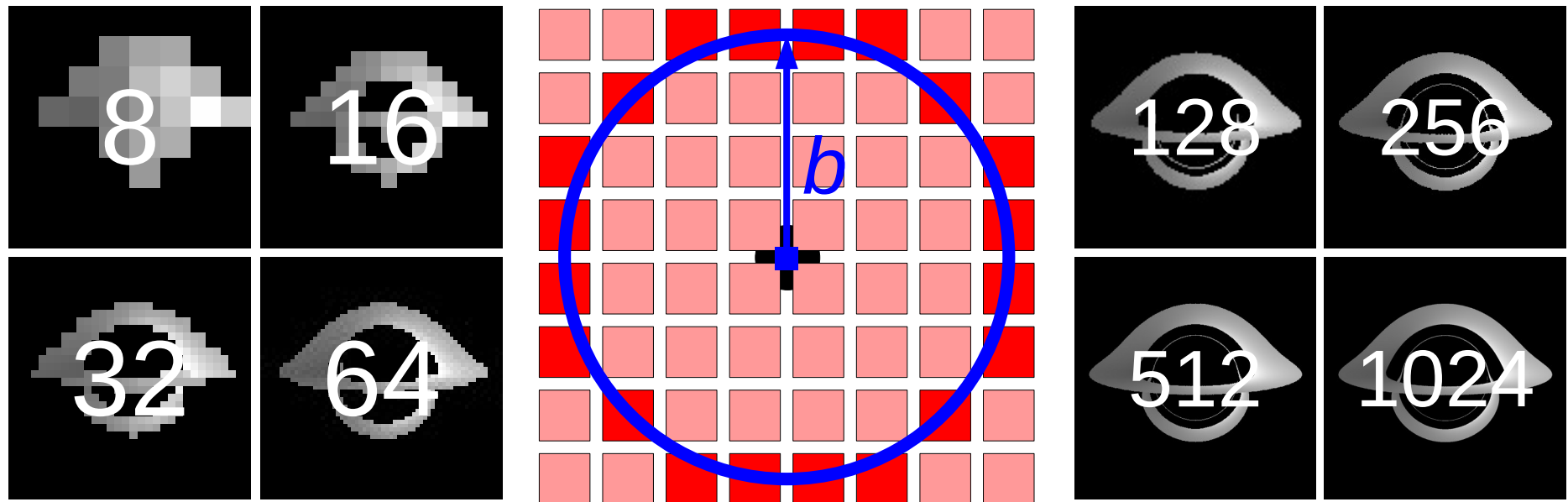
La méthode : « lancer des rayons » de l'oeil au disque de plasma

- Pour chaque pixel de l'image
 - Calculer la trajectoire (résoudre le système d'équations)
 - Regarder si le photon intercepte le disque
 - Si la distance du photon inférieure au rayon de Schwarzschild
 - Dommage... (en même temps, c'est le principe du « trou noir »)
 - Si le photon traverse le plan du disque entre ses rayons intérieur & extérieur
 - Estimation de l'effet Doppler & Einstein
 - Estimation du flux par deux méthodes :
 - Émission monochromatique : simple mais instructive
 - Émission de « corps noir » : plus réaliste mais spectre de Planck
- Méthode systématique mais très coûteuse :
 - Aucune exploitation de la symétrie du problème physique

Méthode « économique » : exploitation symétrie cylindrique

- Pour chaque « paramètre d'impact » (distance au centre)
 - Calcul de la trajectoire du photon en fonction de l'angle
 - Pour chacun des pixels de l'image avec ce paramètre d'impact :
 - Estimation de l'indice d'interception correspondant à l'angle du disque
 - Test si la distance au centre pour cet indice est entre les rayons interne et externe
 - Estimation de l'effet Doppler & Einstein
 - Estimation du flux par deux méthodes :
 - Émission monochromatique : simple mais instructive
 - Émission de « corps noir » : plus réaliste mais spectre de Planck
- Beaucoup plus efficace et temps de calcul $\sim \# \text{pixels}$
 - Exploitation du PixHertz (nombre de pixels sur temps écoulé)...

Du paramètre d'impact « b » Au cercle sur l'image



Balayage des pixels : découpage du cercle en $8b$ secteurs

Le code, la boucle principale : paramètres d'impact & angles

```
for (n=1;n<=nmx;n++)
{
    h=4.*PI/(MYFLOAT)TRACKPOINTS;
    d=stp*n;
    db=bmx/(MYFLOAT)nmx;
    b=db*(MYFLOAT)n;
    up=0.;
    vp=1.;
    pp=0.;
    nh=1;
    rungekutta(&ps,&us,&vs,pp,up,vp,h,m,b);
    rp[(int)nh]=fabs(b/us);
    do
    {
        nh++;
        pp=ps;
        up=us;
        vp=vs;
        rungekutta(&ps,&us,&vs,pp,up,vp,h,m,b);
        rp[(int)nh]=b/us;
    } while ((rp[(int)nh]>=rs)&&(rp[(int)nh]<=rp[1]));
    for (i=nh+1;i<TRACKPOINTS;i++)
    {
        rp[i]=0.;
    }
}
```

```
imx=(int)(8*d);
for (i=0;i<=imx;i++)
{
    phi=2.*PI/(MYFLOAT)imx*(MYFLOAT)i;
    phd=atanp(cos(phi)*sin(tho),cos(tho));
    phd=fmod(phd,PI);
    ii=0;
    tst=0;
    do
    {
        php=phd+(MYFLOAT)ii*PI;
        nr=php/h;
        ni=(int)nr;
        if ((MYFLOAT)ni<nh)
        {
            r=(rp[ni+1]-rp[ni])*(nr-ni*1.)+rp[ni];
        }
        else
        {
            r=rp[ni];
        }
        if ((r<=re)&&(r>=ri))
        {
            tst=1;
            impact(d,phi,dim,r,b,tho,m,zp,fp,q,db,h,bss,raie);
        }
        ii++;
    } while ((ii<=2)&&(tst==0));
}
```

Centre Blaise Pascal et son centre d'essais...

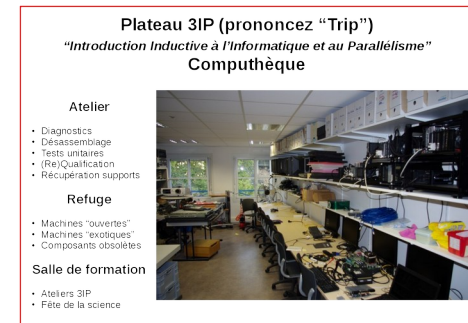
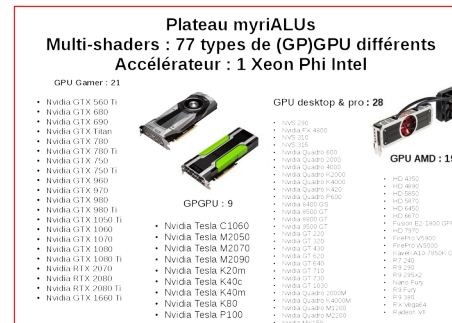
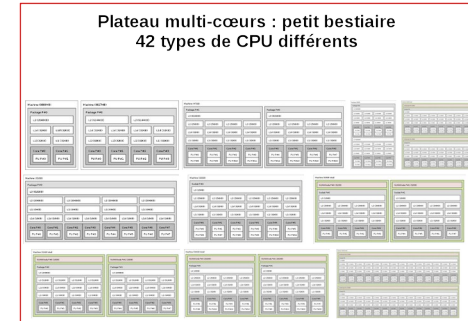
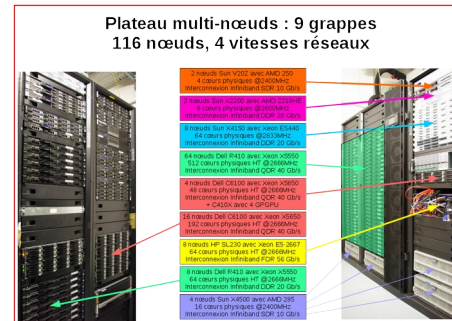
- Centre Blaise Pascal : 3 hébergements

- Hôtels à conférences
- Hôtel à formations
- Hôtel à projets

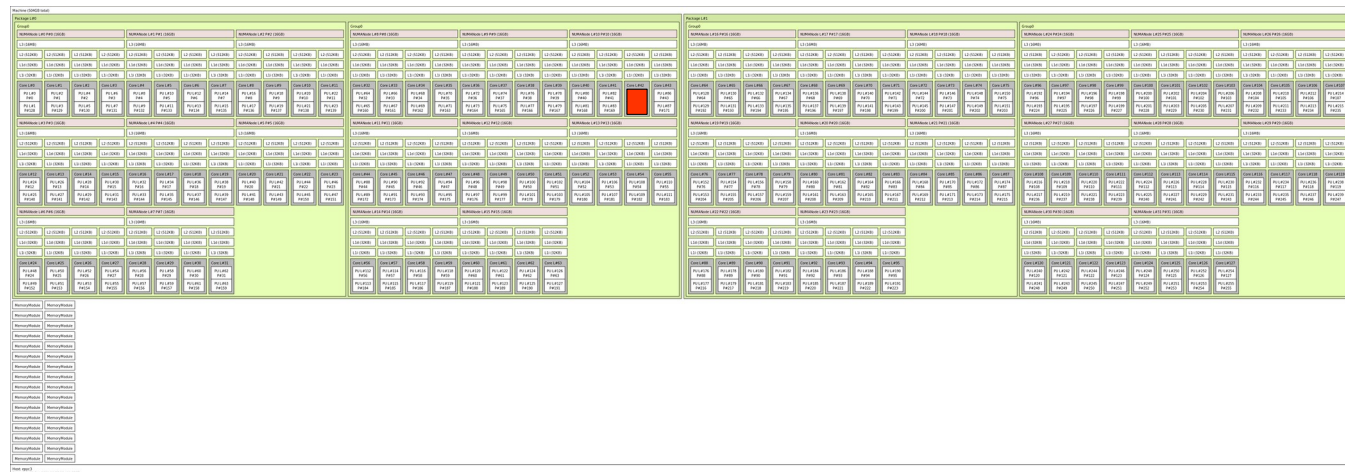
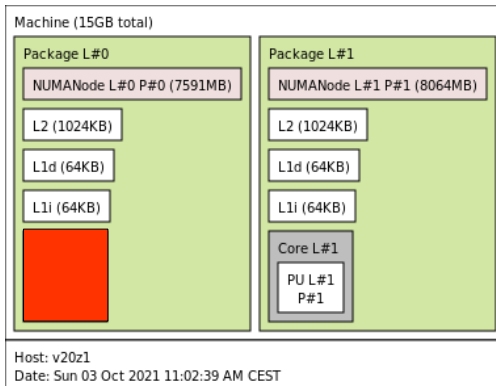
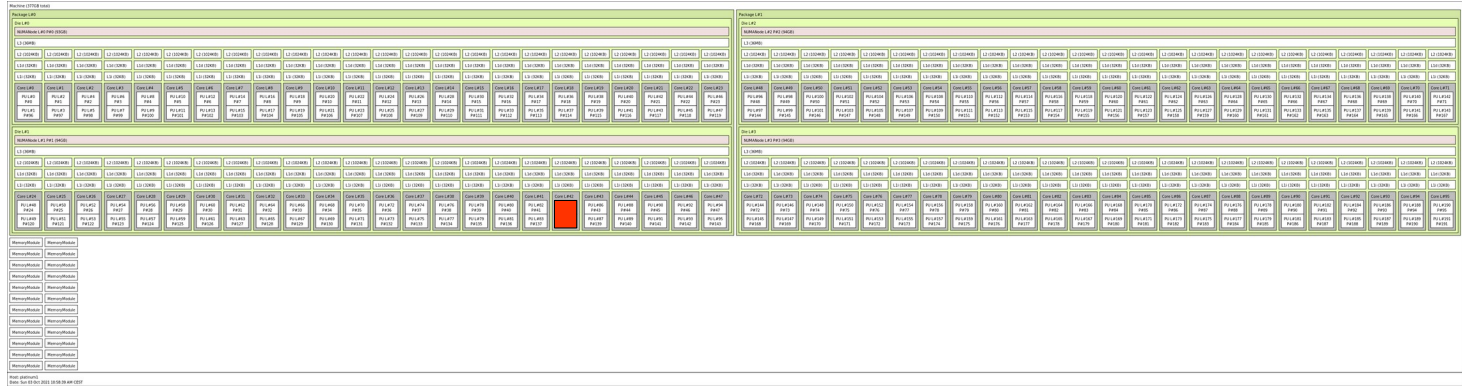
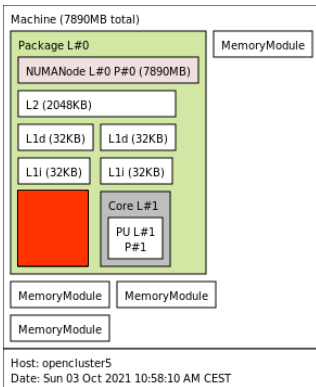
- Centre d'essais : 3 quêtes

- Reproductibilité
- Scalabilité
- Simplicité

- Ses propres plateaux techniques



Plateau multi-cœurs : 280 hôtes de 2 à 128 cœurs : les extrêmes



Plateau multi-shaders : (GP)GPU

108 modèles différents...

dont 72 accessibles **directement** !



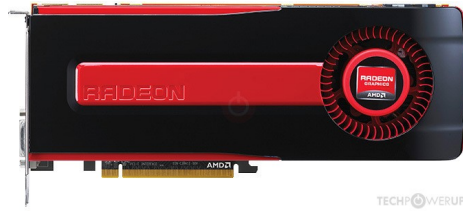
GPU Gamer :
33 modèles
De la GT 640
... à la RTX 3090



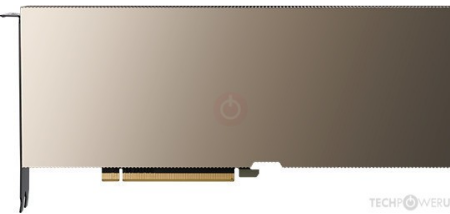
GPGPU :
15 modèles
Nvidia Tesla C1060
... à la Nvidia A100



GPU desktop & pro :
34 modèles
De la Quadro FX 4800
... à la RTX 6000



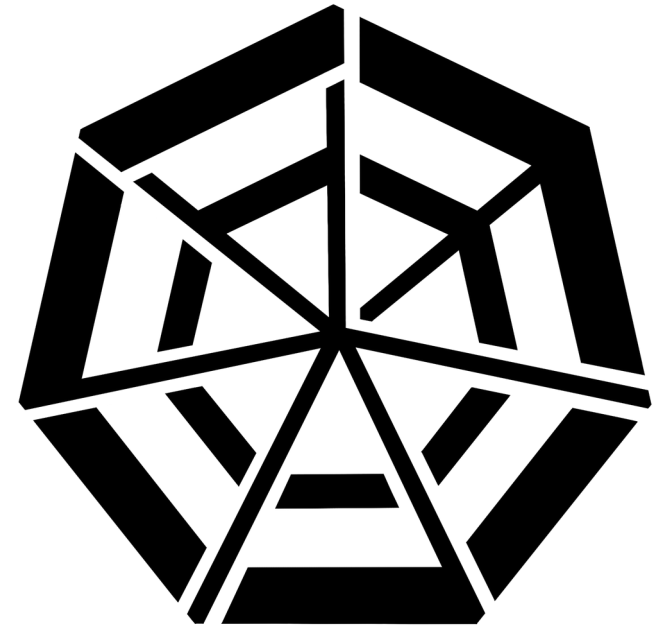
GPU AMD Gamer :
26 modèles
De la HD 7970
... à la RX 6900 XT



Sur les Machines du CBP : SIDUS

Je n'installe pas, je démarre !

- **Quoi ?**
 - Déployer un système simplement sur un parc de machines
- **Pourquoi ?**
 - Assurer l'unicité des configurations
 - Limiter l'empreinte du système sur les disques
- **Pour qui ?**
 - Étudiants (vous quoi!), enseignants, chercheurs, ingénieurs, ...
- **Quand & Où ?**
 - Centre Blaise Pascal : depuis 2010, plus de 280 machines
 - PSMN : depuis 2011, plus de ~800 nœuds (sa propre instance)
- **Comment ?**
 - Utiliser un partage en réseau d'une arborescence
 - Détourner une ruse de LiveCD



« Deux machines ayant démarré SIDUS ne peuvent pas ne pas avoir le même système ! »

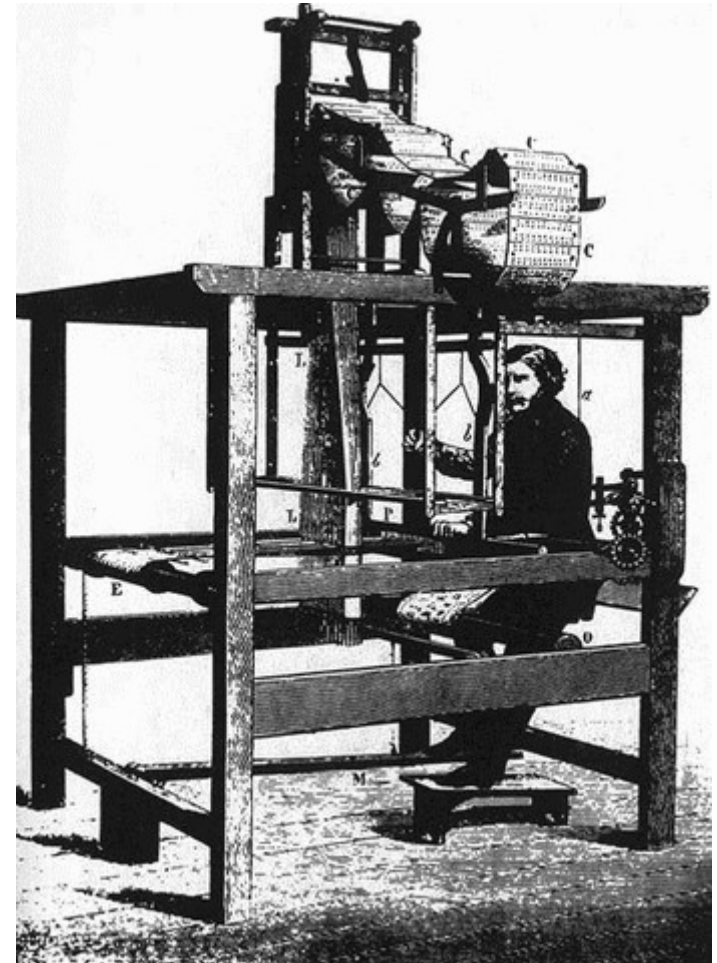
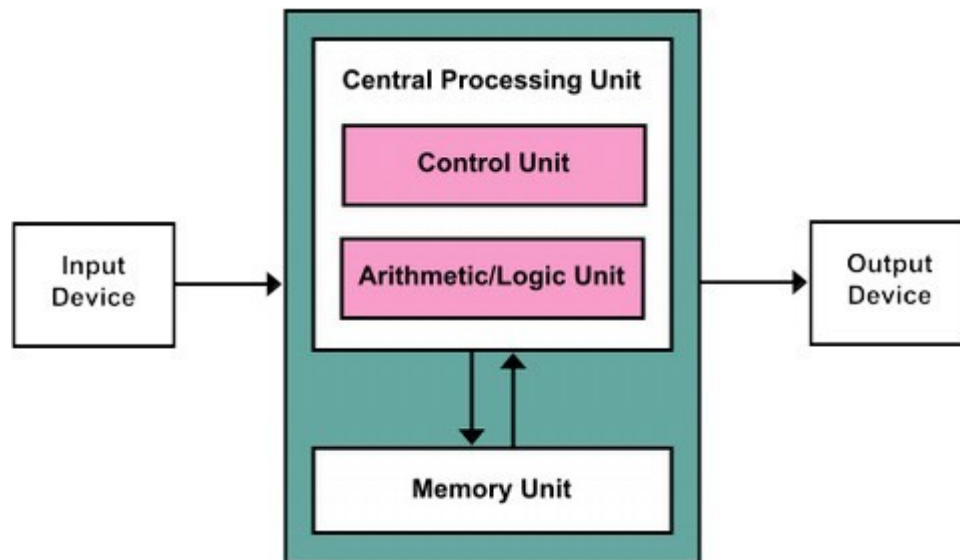
Computhèque

40 ans d'histoire de l'informatique

- Ordinateurs :
 - Thomson TO8, Amiga 500, MacSE, ...
- Processeurs (et leurs cartes-mère) :
 - 80386SX, 80486SX, Overdrive DX4, Amd5x86, K6-2, K7, ...
- Périphériques de stockage :
 - Cartes, câbles, disques, lecteurs de bande, ... en SCSI, Firewire, FC
- Périphériques de communication :
 - Ethernet 10Base2, 10Base-T, ATM, Myrinet, Infiniband, ...
- OS : natifs ou Debian Buzz, Hamm, Squeeze, ...

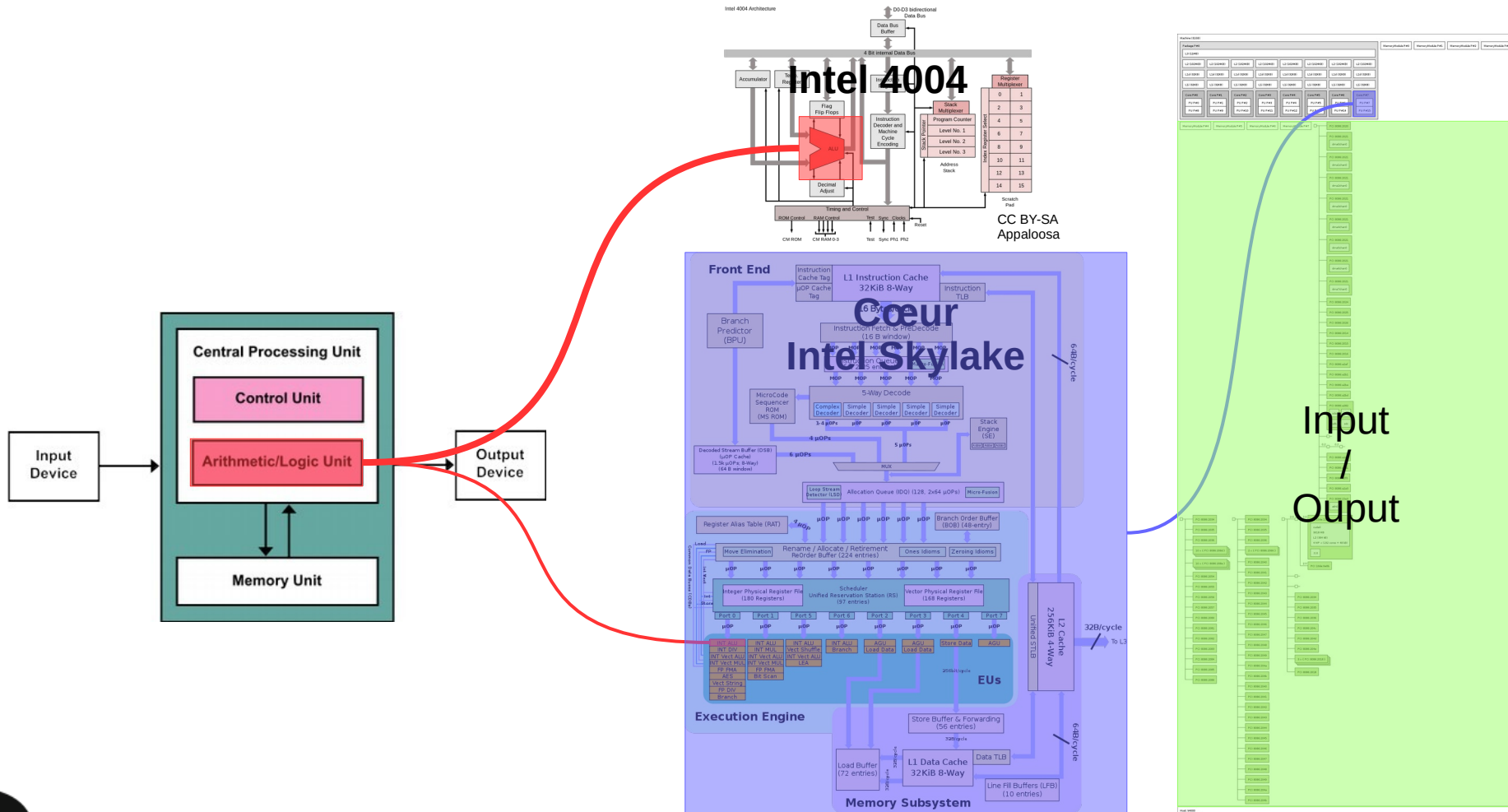
Un ordinateur : un métier à tisser ?

Architecture de Von Neumann



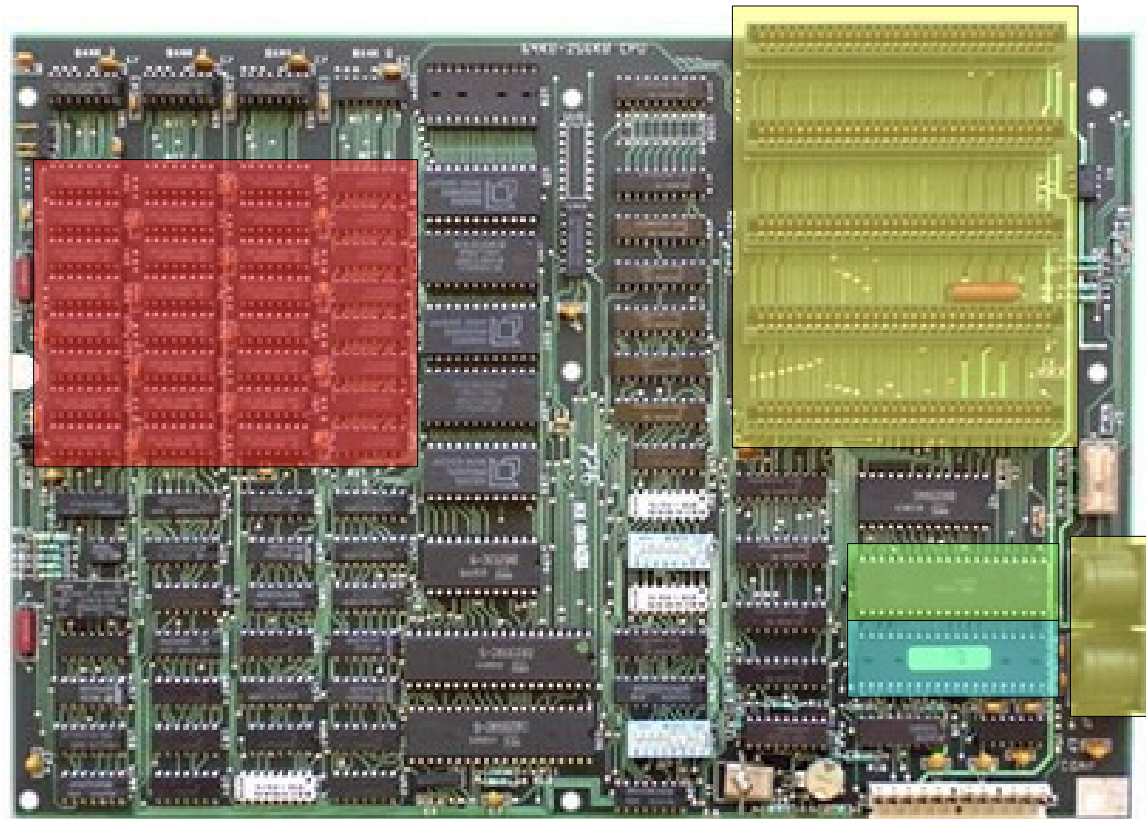
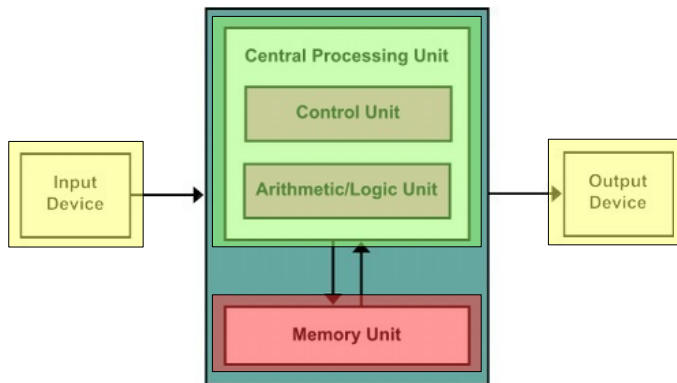
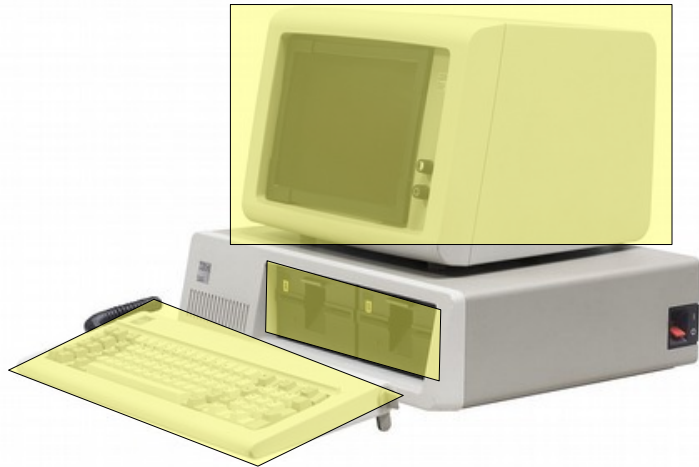
Architecture de Von Newman

50 ans d'évolution chez Intel

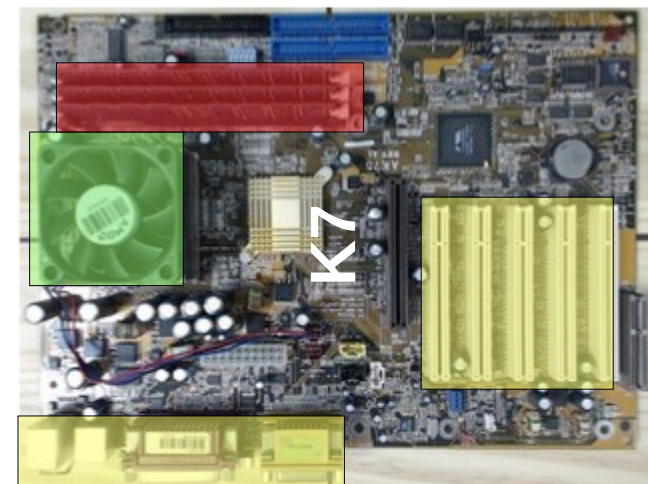
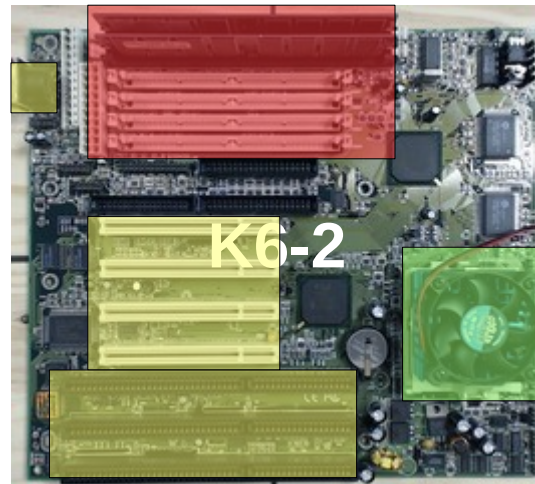
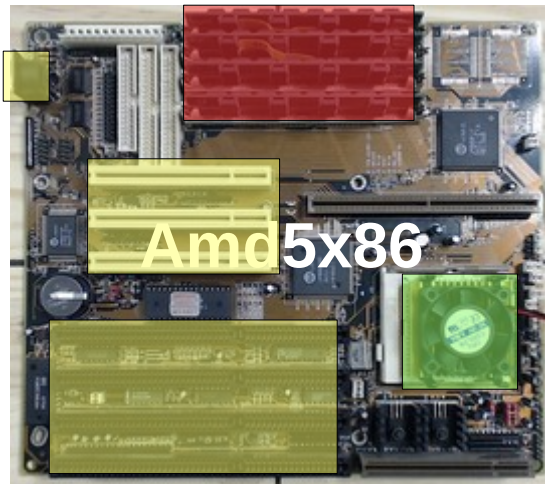
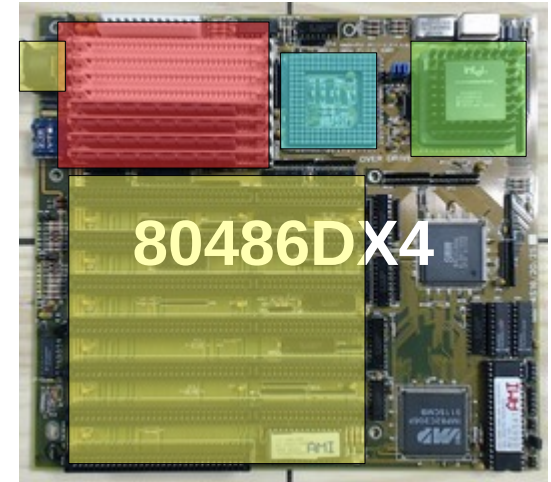


Une révolution informatique

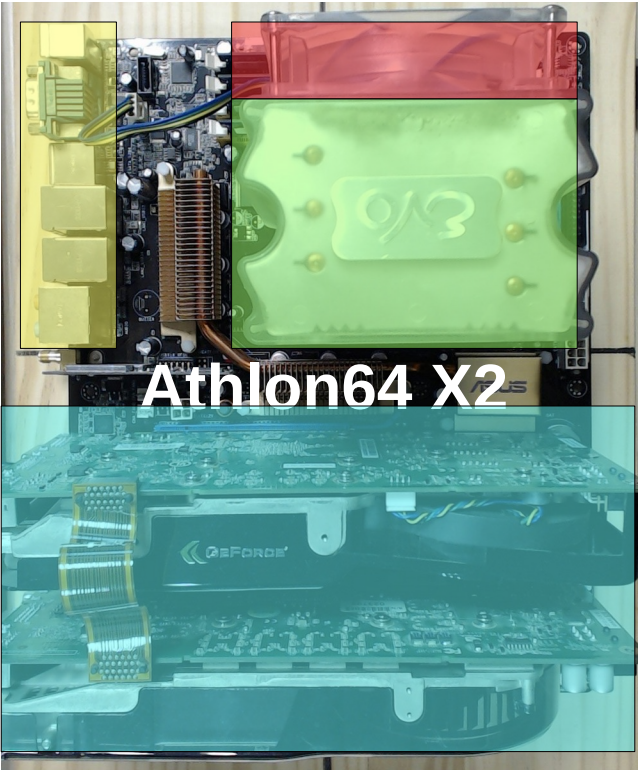
IBM Personal Computer, le « PC »



Evolution des cartes mères en un clin d'œil, de 1989 à 2002



Evolution des cartes mères en un clin d'œil, de 2005 à 2018



Banc de test

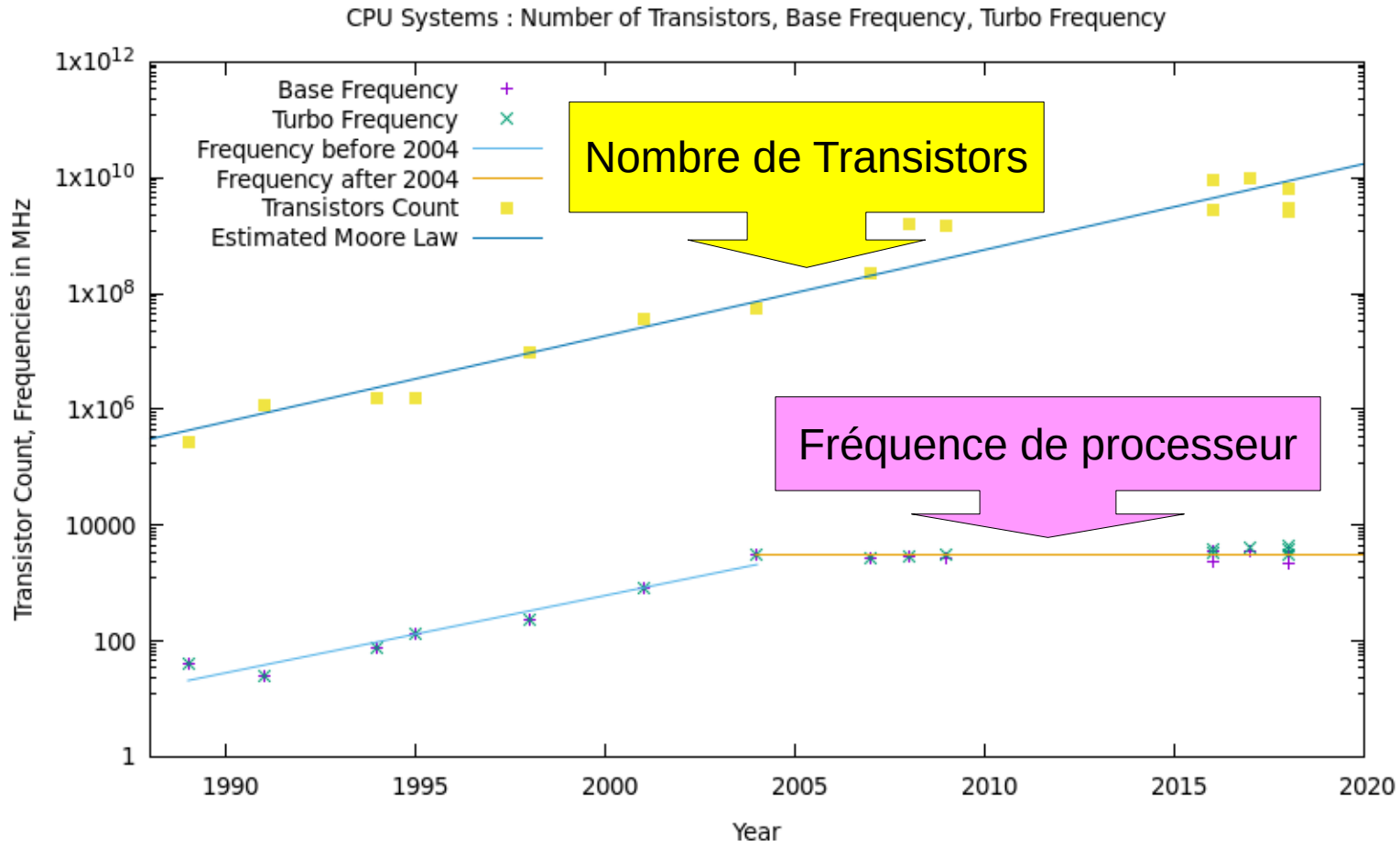
Entre le code et le matériel, l'OS

- Une même distribution de 1996 à 2019 : Debian (presque)
 - Debian Buzz : 80386SX
 - Debian Hamm : 80486SX, 80486DX4, Amd5x86, K6-2
 - Debian Stretch 32 bits : K7, Northwood
 - Debian Stretch 64 bits : les 8 autres
 - Ubuntu 18.10 64 bits : TR 1950X
- Composants nécessaires :
 - Programme séquentiel : compilateur C
 - Programme OpenMP : compilateur C et librairie OpenMP
 - Programme OpenCL : interpréteur Python & PyOpenCL, pilotes Nvidia & ROCM
 - Programme CUDA : interpréteur Python & PyCUDA, pilotes Nvidia



Distribution de CPU pertinente ?

Transistors & Fréquences...

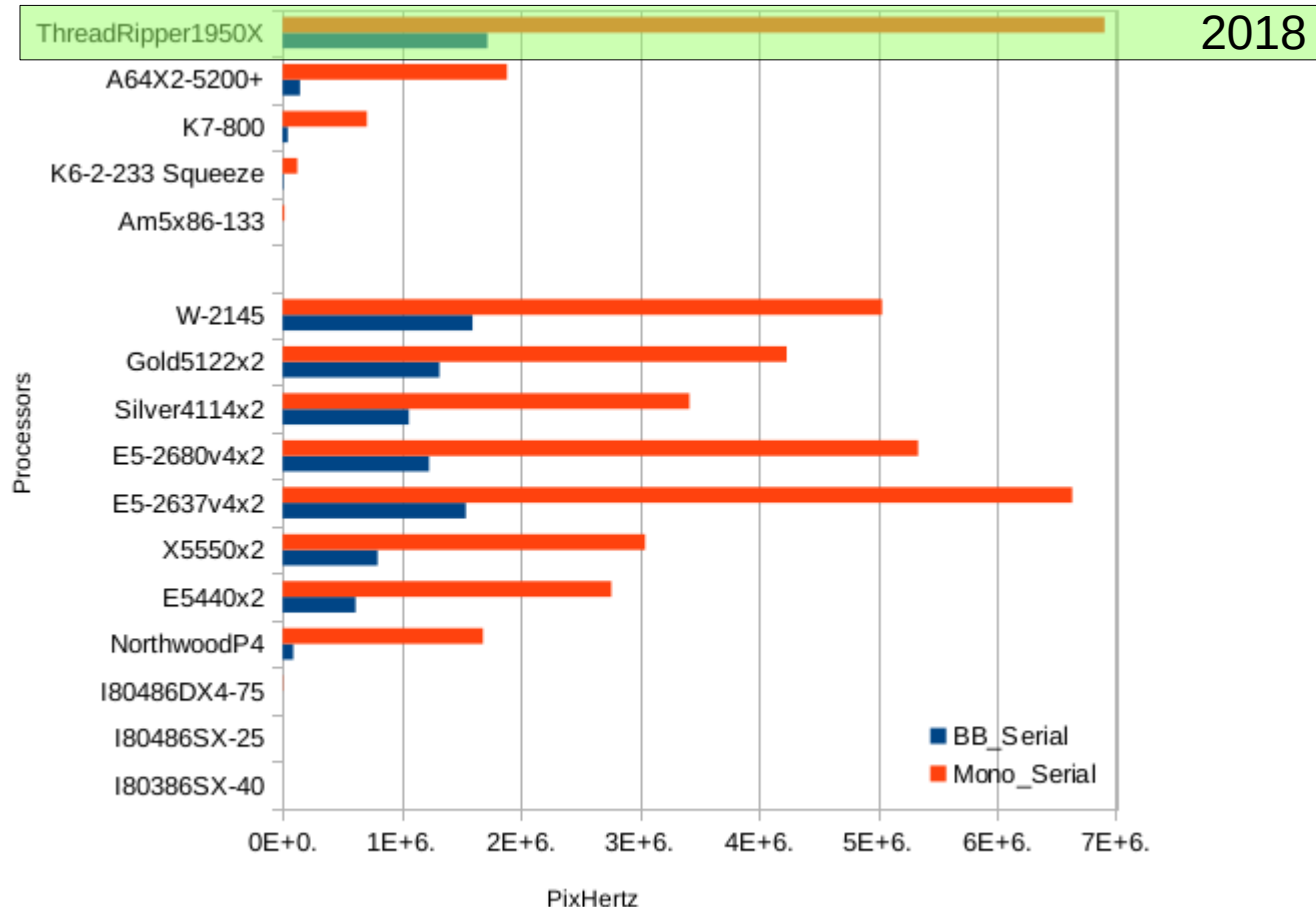


Doublement des transistors tous les 2 ans...

Banc de test sur les 16 CPUs

- Les processeurs et leurs distributions :
 - 80386SX, 80486SX, Overdrive DX4, Amd5x86 : Debian Buzz & Hamm
 - K7, Northwood, AthlonX2 : Debian Stretch
 - E5440x2, X5550x2, E5-2637v4x2, E5-2680v4, Gold5122, Silver4144, W-2145 : Debian Stretch
 - Threadripper 1950X : Ubuntu 18.10
- Images de 64x64 à 16384x16384 pixels : 2^6 à 2^{14}
 - Sauf pour les très très vieux CPU : limitation à 256x256
- Méthodes : 2 à explorer avec « charges » différentes
 - Charge calculatoire faible : « Monochromatique » (ak Mono)
 - Charge calculatoire élevée : « Corps Noir » (aka BB)
- Statistiques : 10 lancements successifs
 - Exploitation de la médiane pour le « Elapsed Time »

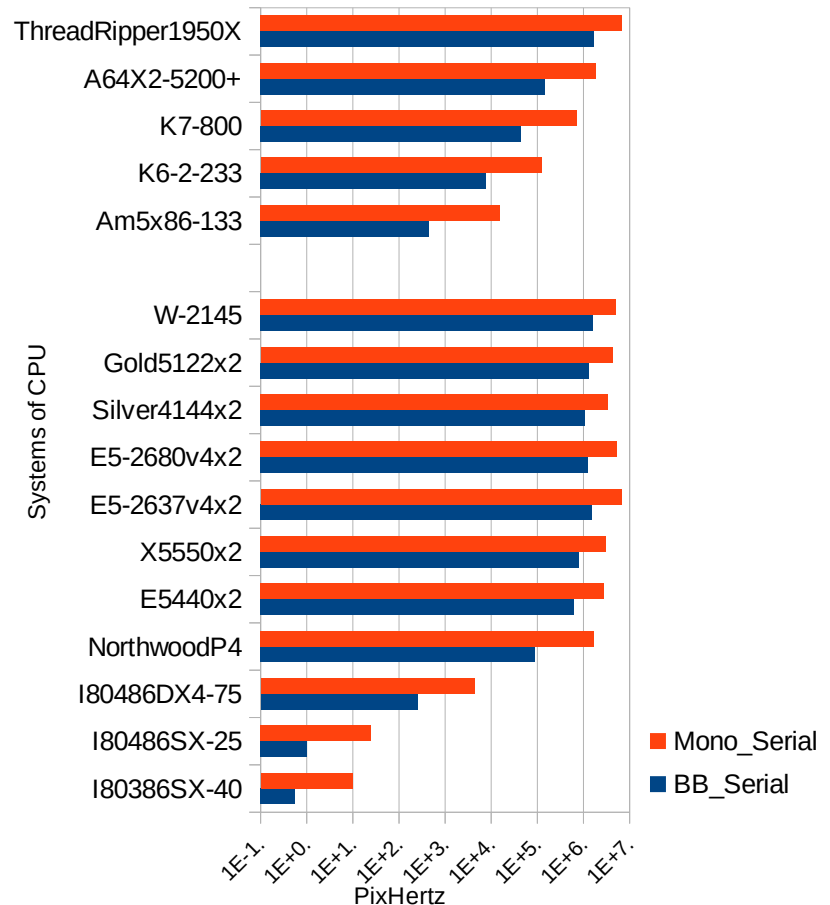
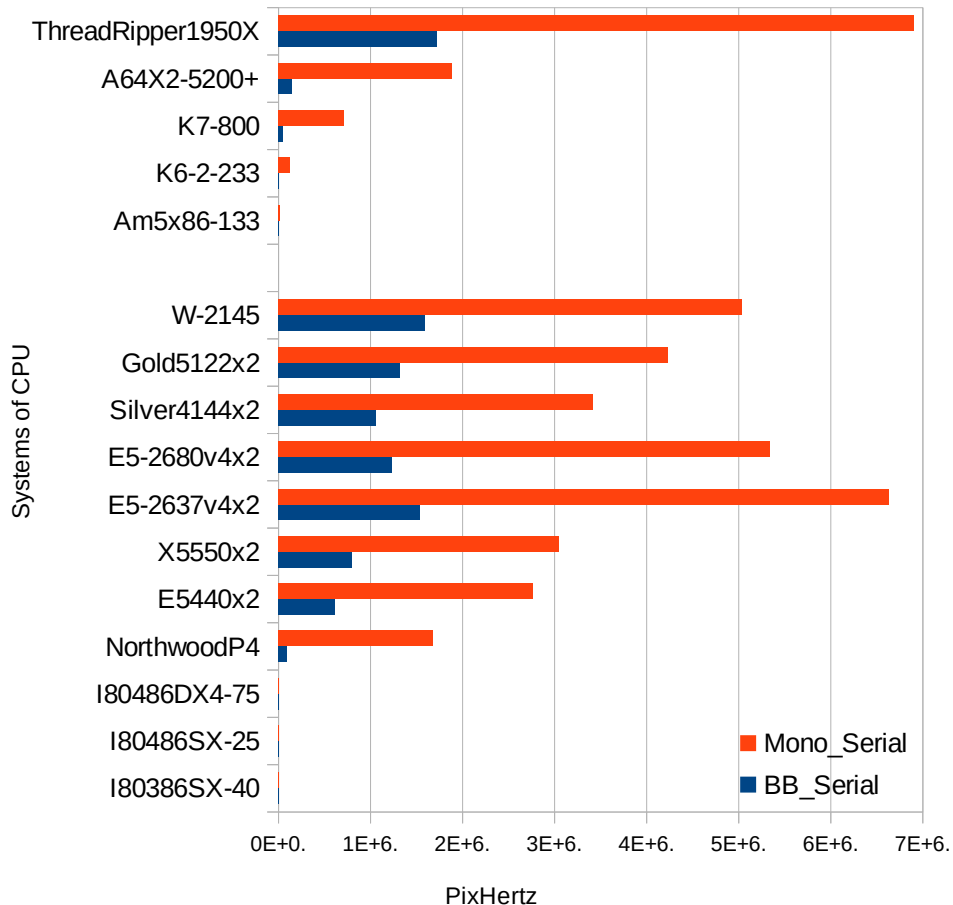
16 processeurs de 1989 à 2019 : « *and the winner is* »...



Le AMD Threadripper 1950X à 3.6 GHz

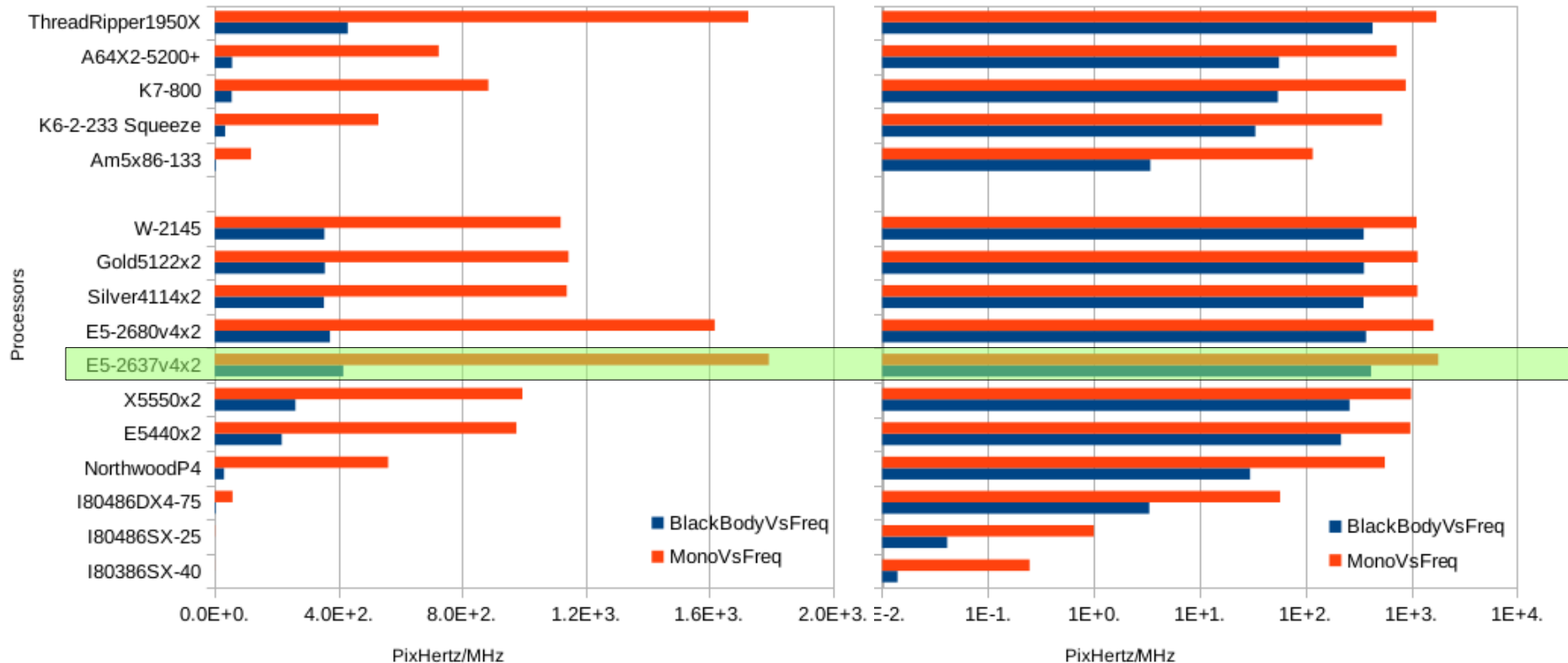
Exécution du code séquentiel

On gagne (quand même) en 30 ans



Gain Best/Worse : 3 millions en BB et 700000 en Mono...

Influence de la fréquence : de 1989 à 2019 : de 40 à 3700 MHz



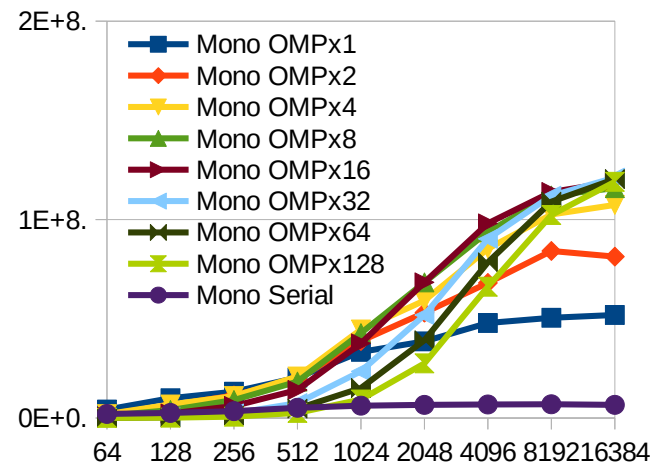
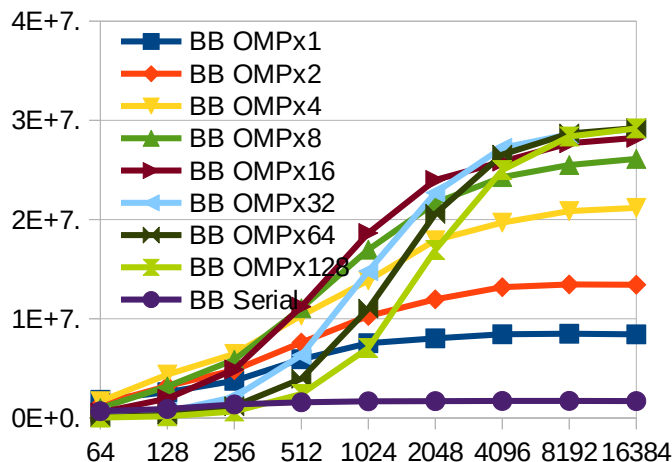
Sur 15 ans, entre un facteur 3 et un facteur 15...

Il va falloir trouver autre chose pour accélérer !

Parallélisation du code

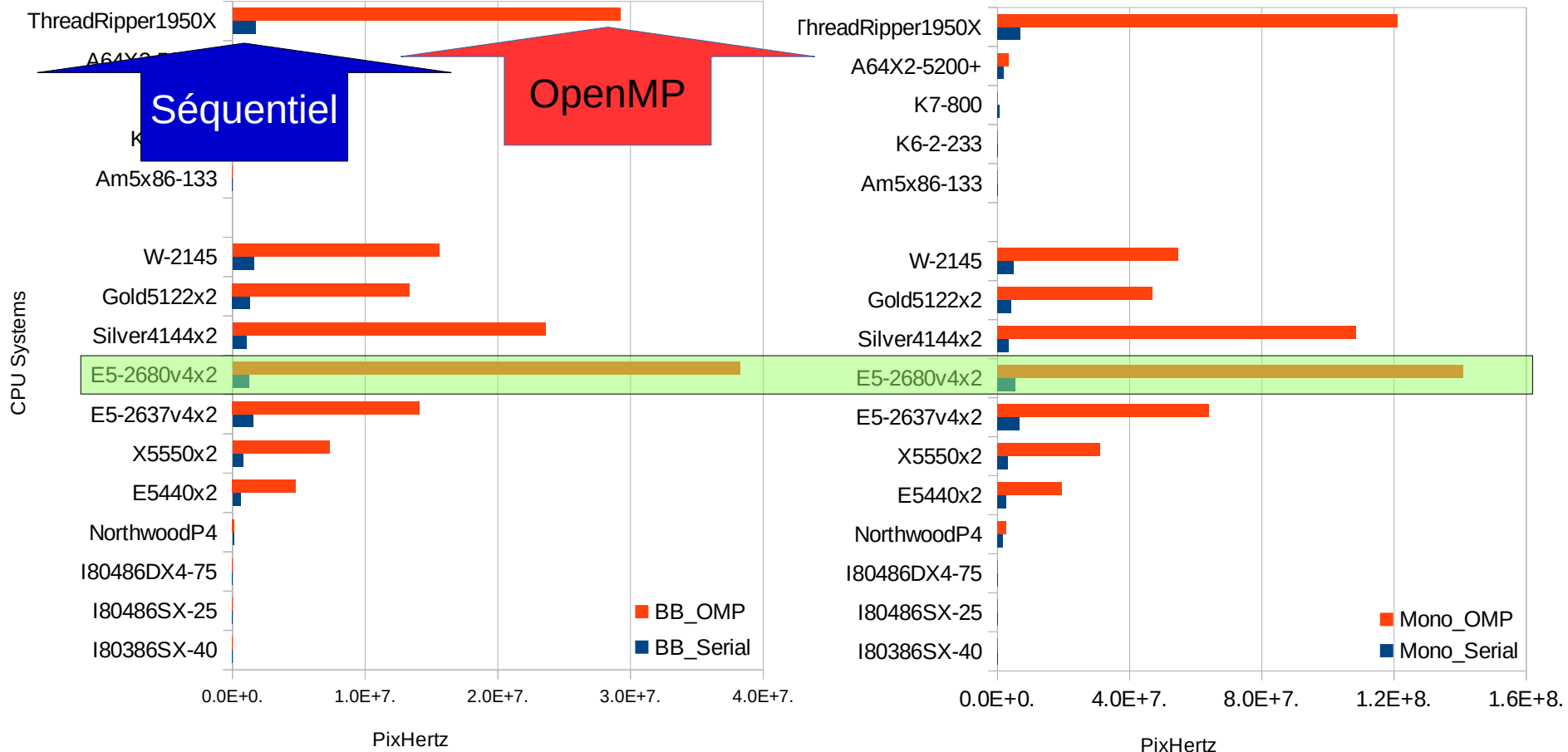
Passage en OpenMP & étranger

- Parallélisation « naturelle » : paramètre d'impact
 - Modification mineure du code (déplacement de déclaration de variables)
 - Crainte : charge calculatoire non équivalente pour les différents tâches
 - Exploration pour différents OMP_NUM_THREADS : de 1x à 128x
- Pour le meilleur : le ThreadRipper 1950x, un **x17-x18**



Parallélisation OpenMP

On gagne plus que prévu en BB !



x68 millions en BB et x14 millions en Mono

Peut-on mieux faire ?

Testons OpenCL !

- OpenCL, méconnu mais tellement polyvalent : 13 implémentations
 - GPU : Nvidia, AMD via ROCm, AMD via Mesa, Intel via Beignet et Intel
 - CPU : AMD, PortableCL, **Intel**
 - MIC : Intel pour Xeon Phi
 - FPGA : Altera/Intel, Xilinx
 - (DSP : Texas Instruments)
 - (GPU ARM)
- OpenCL : sa programmation...
 - Principe : des « noyaux » de calcul à distribuer à outrance !
 - Programmation « hardcore » en C, plus facile en C++
 - Programmation via API Python : « la voie » !



OpenCL

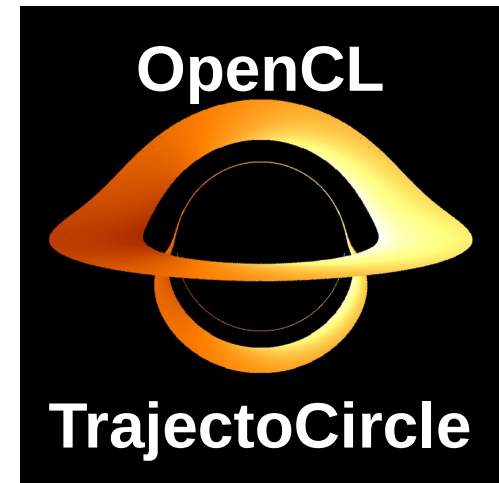
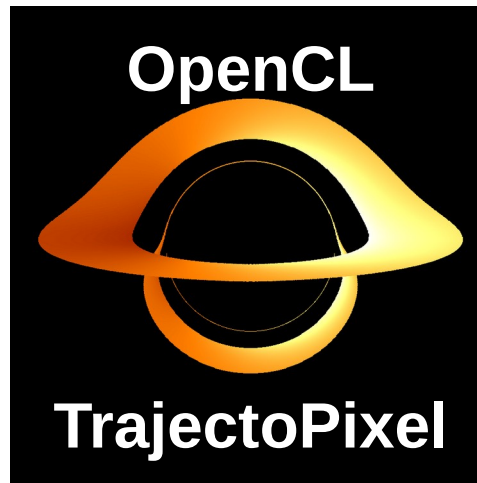
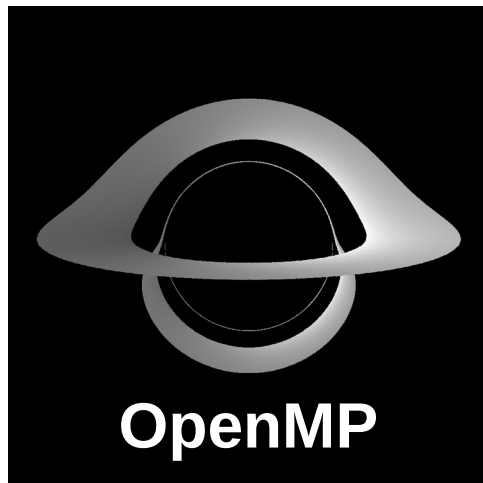
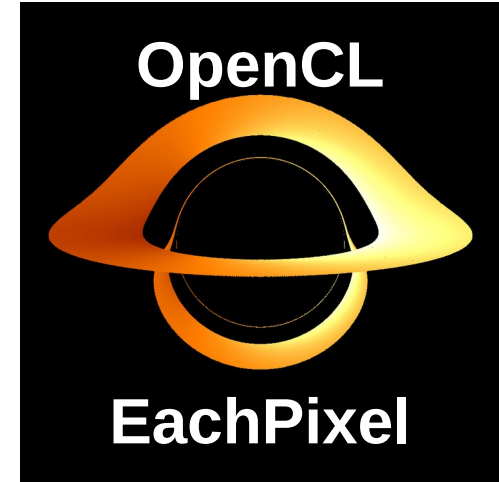
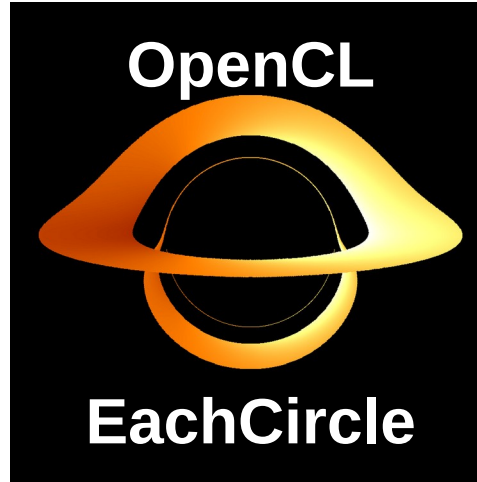
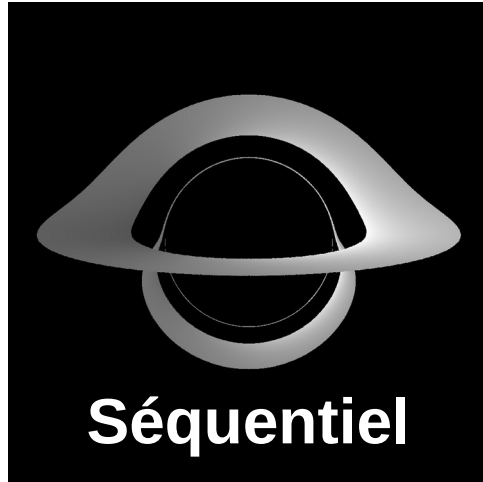
OpenCL : distribuer notre calcul.

Quel régime de parallélisme PR ?

- Approche initiale héritée du code C : **EachCircle**
 - Parallélisé suivant le paramètre d'impact : $PR = Taille/2$
- Approche brutale : **EachPixel**
 - Parallélisé suivant le nombre de pixels : $PR = Taille * Taille$
- Approche hybride : **TrajectoPixel**
 - D'abord parallélisé suivant les paramètres d'impact : $PR = Taille/2$
 - Ensuite parallélisé suivant chaque pixel : $PR = Taille * Taille$
- Approche hybride sauvage : **TrajectoCircle**
 - D'abord parallélisé suivant les paramètres d'impact : $PR = Taille/2$
 - Ensuite parallélisé suivant chaque pixel : $PR = 4 * Taille$
- Donc 4 méthodes à explorer pour tous nos CPU !

Obtient-on les mêmes résultats ?

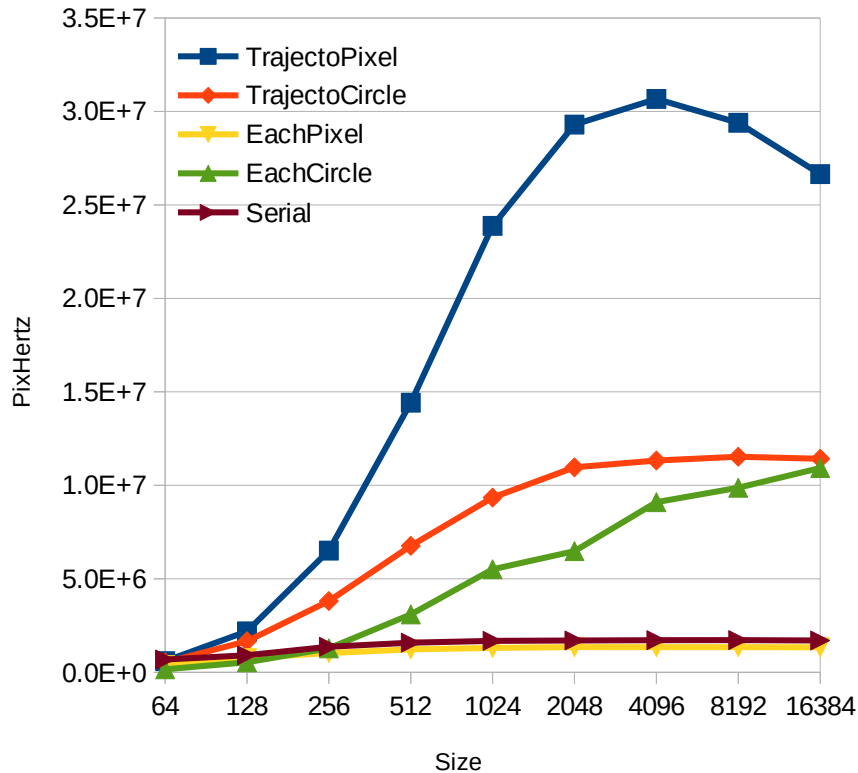
Laissons l'œil juger...



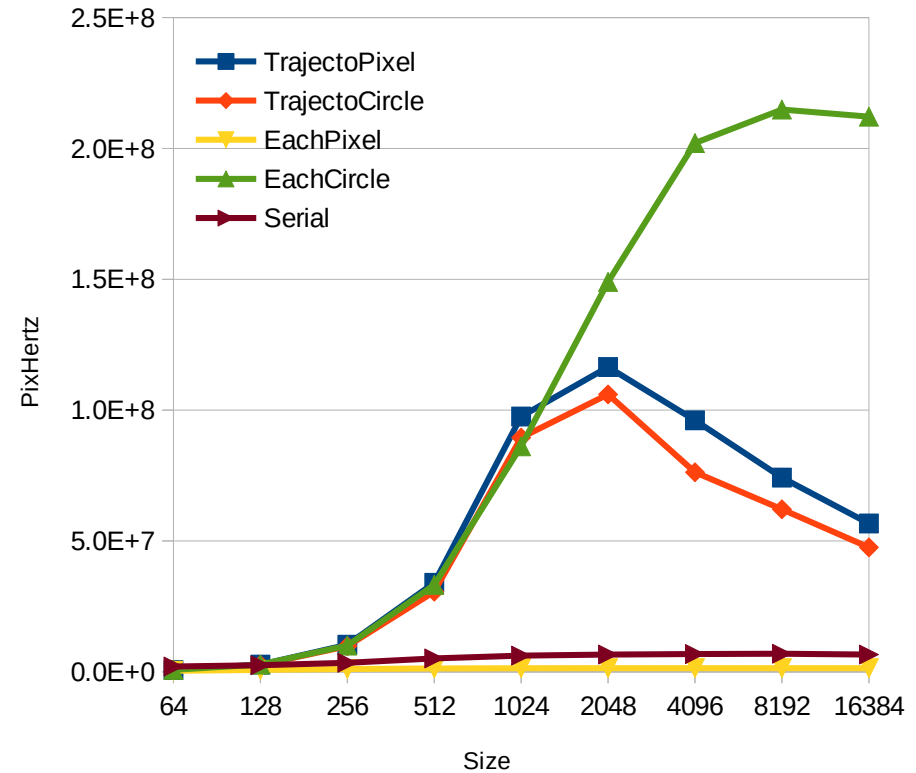
OpenCL sur Threadripper 1950x

La méthode de // importante...

BB on Threadripper 1950X with OpenCL



Mono on Threadripper 1950X

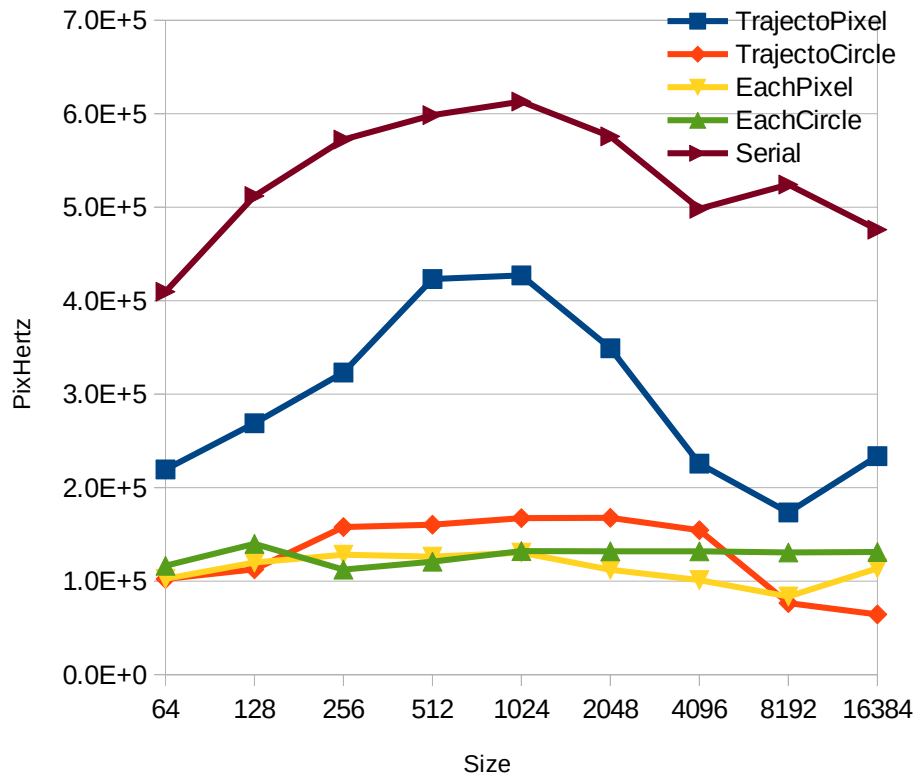


TrajectoPixel pour BB, EachCircle pour Mono...

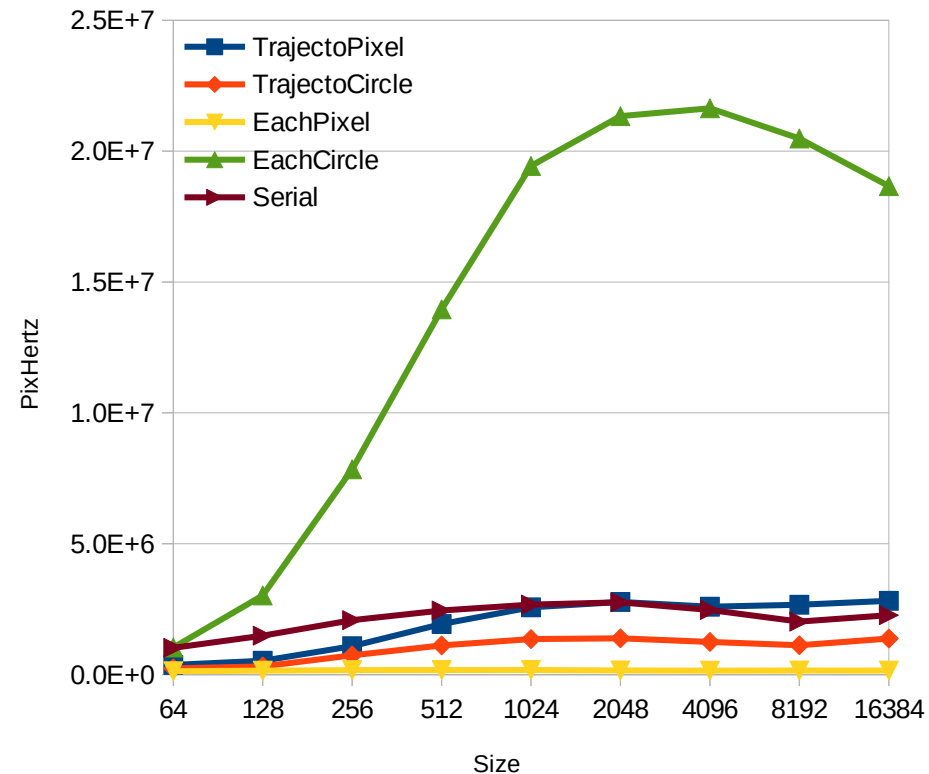
OpenCL sur Harpertown E5440x2

OpenCL (AMD) pas terrible

BB on Harpertown E5440 with OpenCL



Mono on Harpertown E5440 with OpenCL



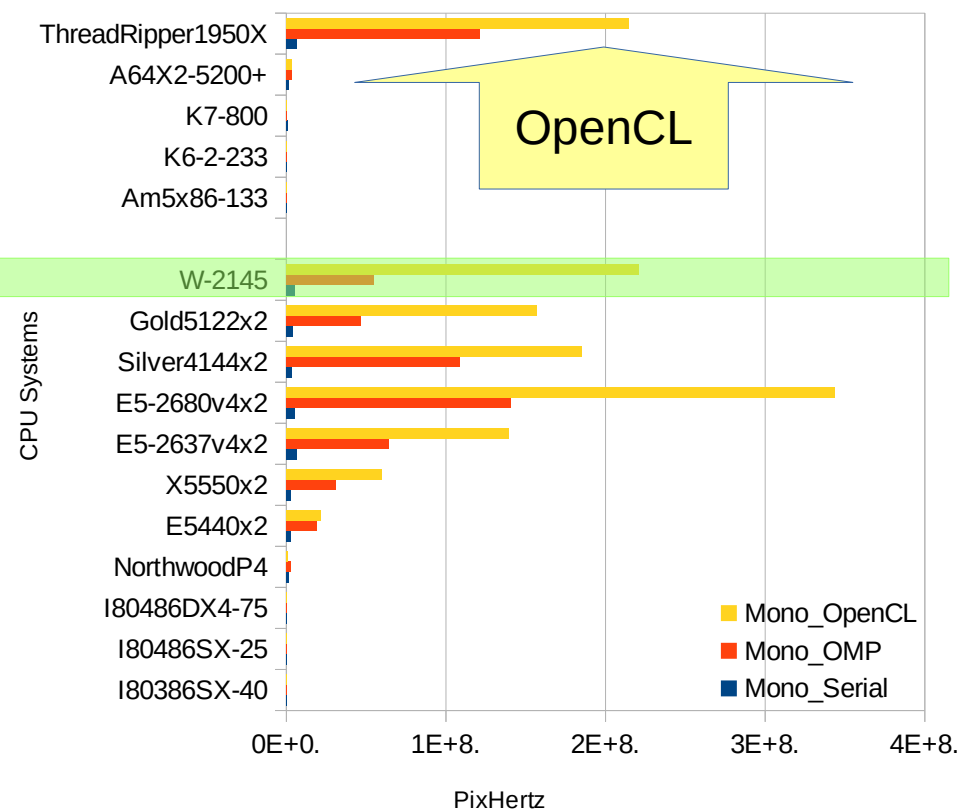
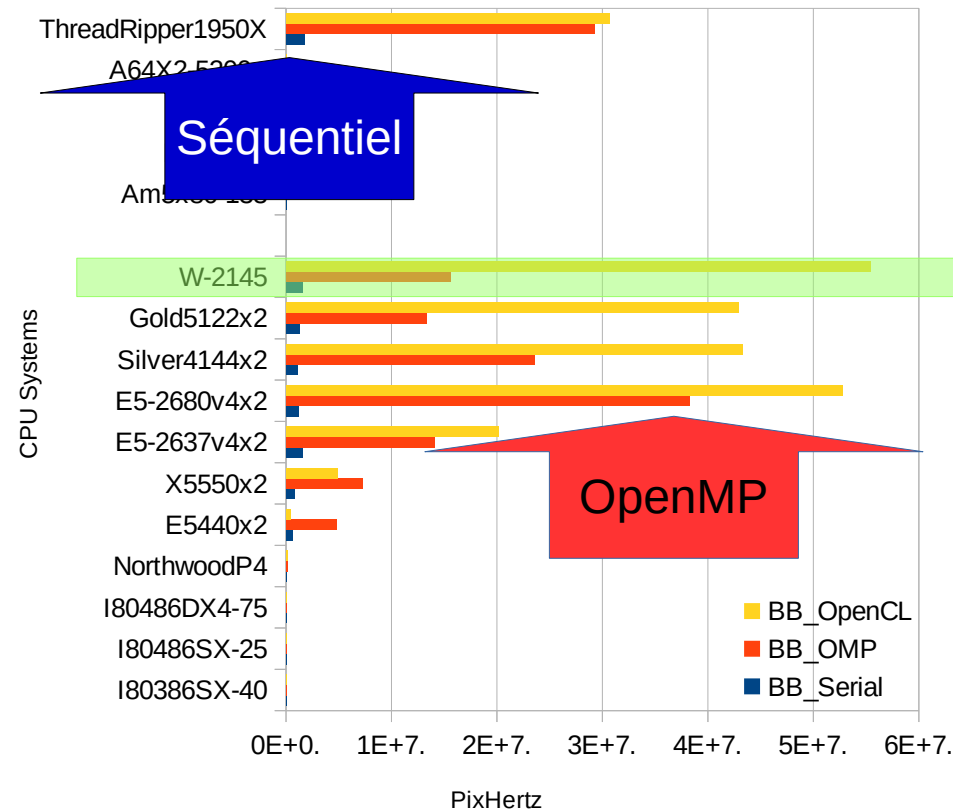
A chaque processeur, sa courbe de scalabilité...

Parallélisation OpenCL

Pour tous les processeurs...

BB with Serial, OpenMP, OpenCL

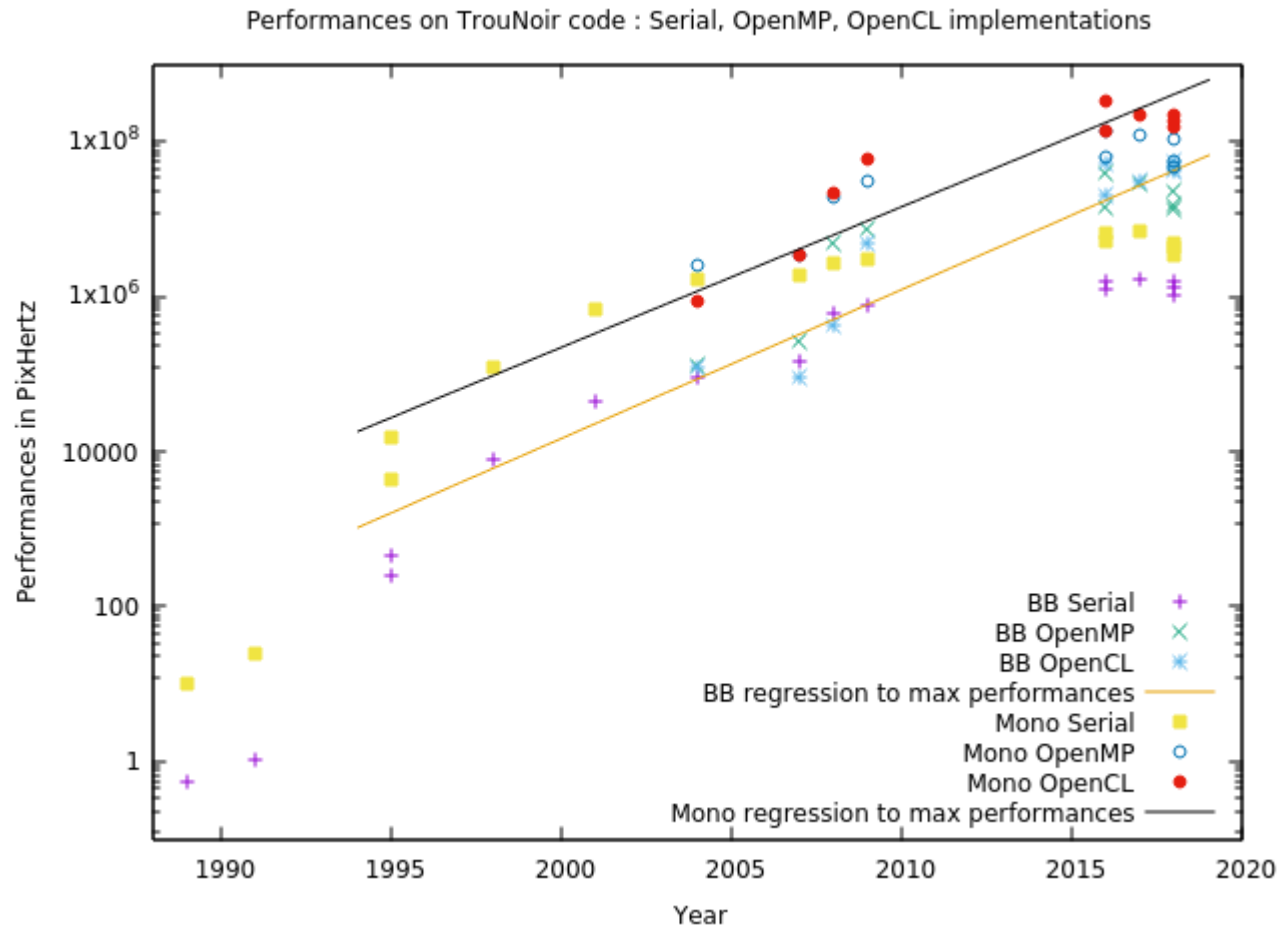
Mono with Serial, OpenMP, OpenCL



OpenCL Intel très efficace avec CPU Intel !

Pour les processeurs

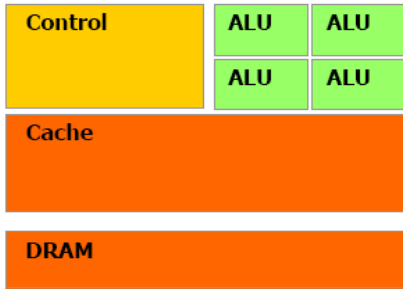
Loi de Moore* respectée ! Enfin...



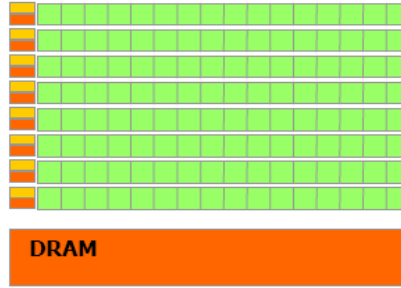
Performance : x2 tous les 18 mois !

Combien de composants à l'intérieur ?

Quelle différence entre GPU & CPU



CPU



GPU

- **Opérations**

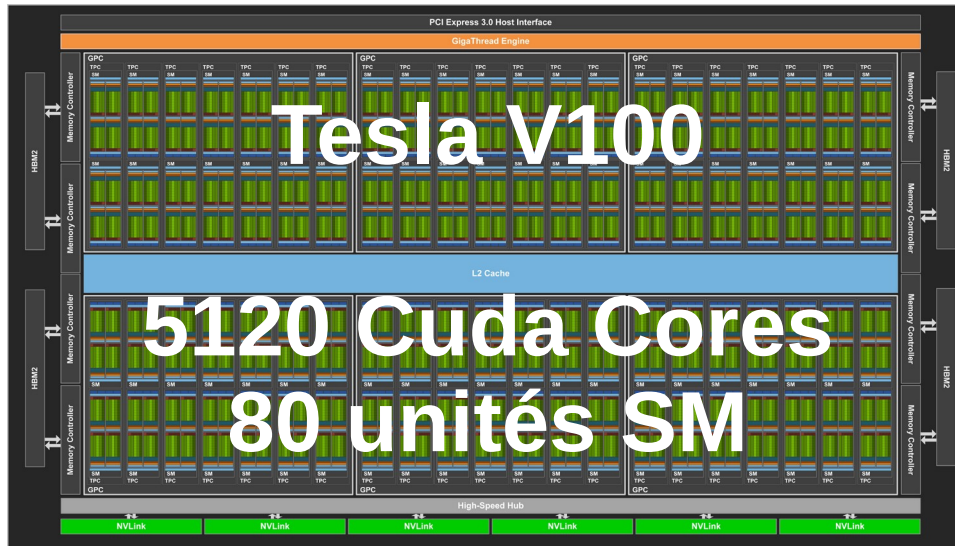
- Multiplication de Matrice
- Vectorisation
- « Pipelining »
- Shader (multi)processeur

- **Programmation : 1993**

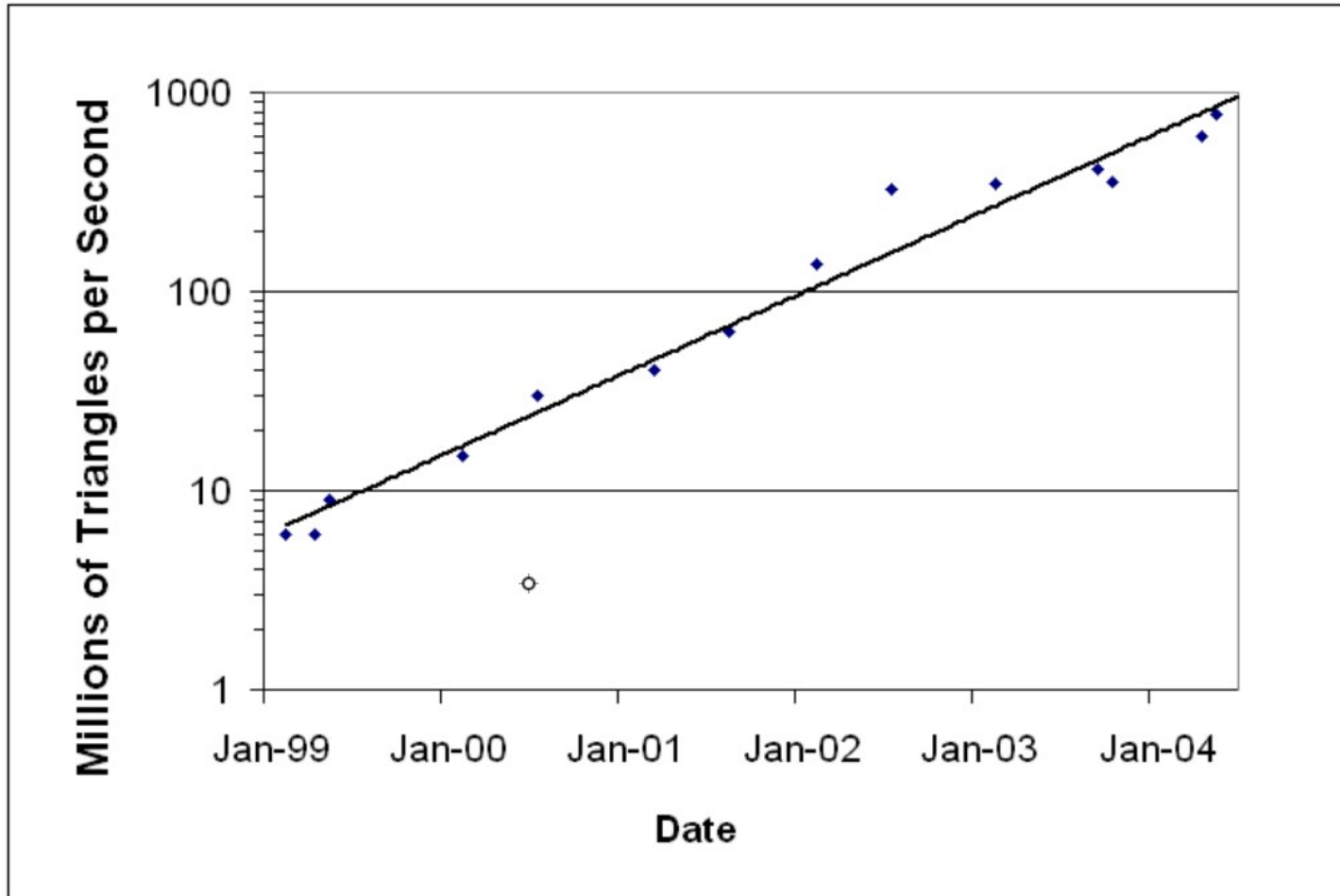
- OpenGL, Glide, Direct3D, ...

- **Généricité : 2002**

- CgToolkit, CUDA, OpenCL



1999-2004 : progression CPU bof... Mais les GPU...



x100 pour les GPUs et seulement x10 pour les CPUs

Ajoutons les GPUs & MIC

Générations Fermi, Kepler, Maxwell, Pascal, Turing

Gamer



GPGPU



Générations GT200, Fermi, Kepler, Pascal, Volta

MIC

Intel : KNC



R9-Nano, Vega 64, Radeon 7

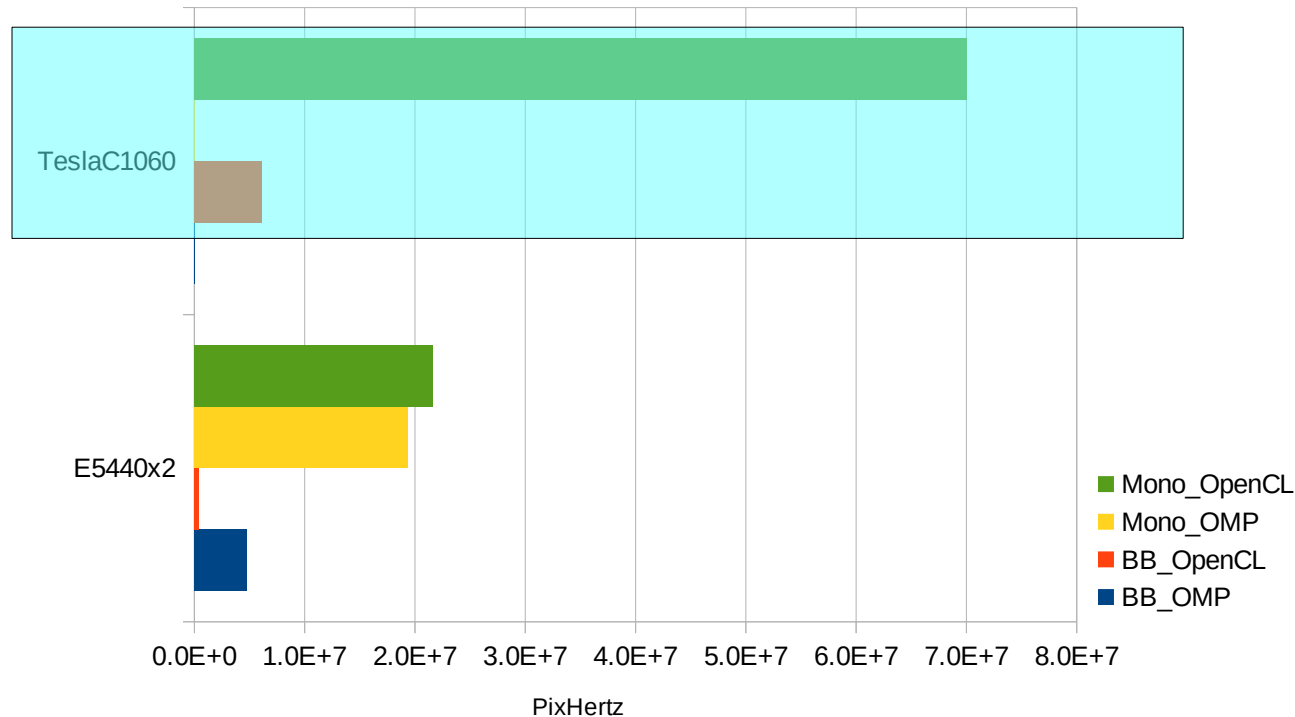


AMD GPU

Le meilleur en computèquede chaque génération...

Comparaison au fil du temps

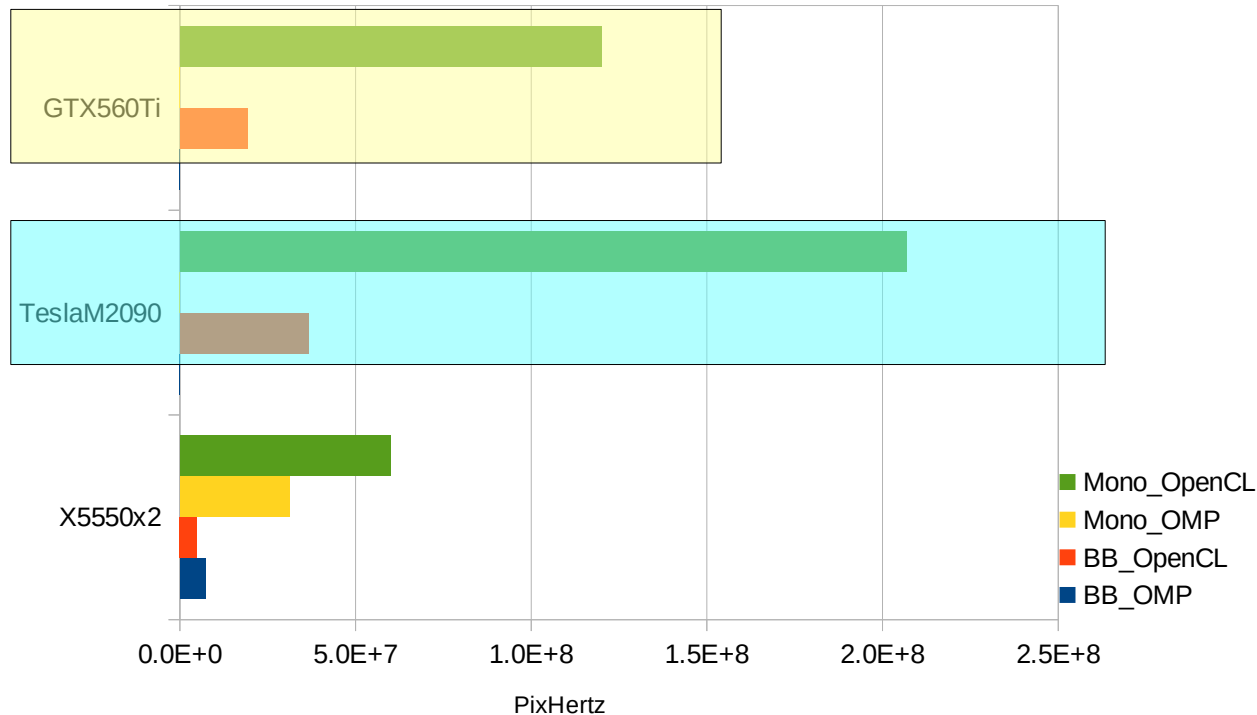
En 2008, E5440 & C1060



- En Mono, Un x3 en OpenCL pour le GPGPU
- En BB, à peine supérieur en OpenCL face à l'OMP

En 2011, entre Nehalem & Fermi

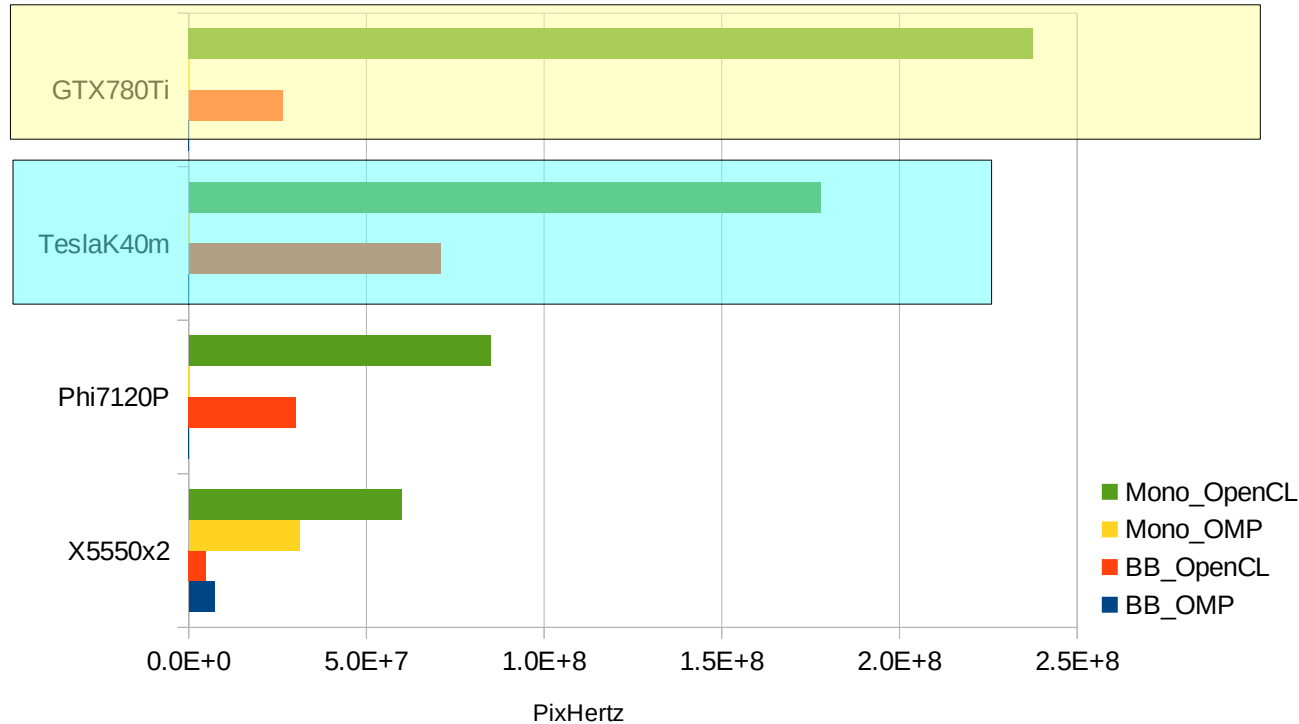
Les différences pas si flagrantes...



- Le GPGPU M2090 gagne d'un facteur 3 en Mono, mais d'un facteur 5 en BB
- La carte de Gamer l'emporte sur le double CPU d'au moins un facteur 2

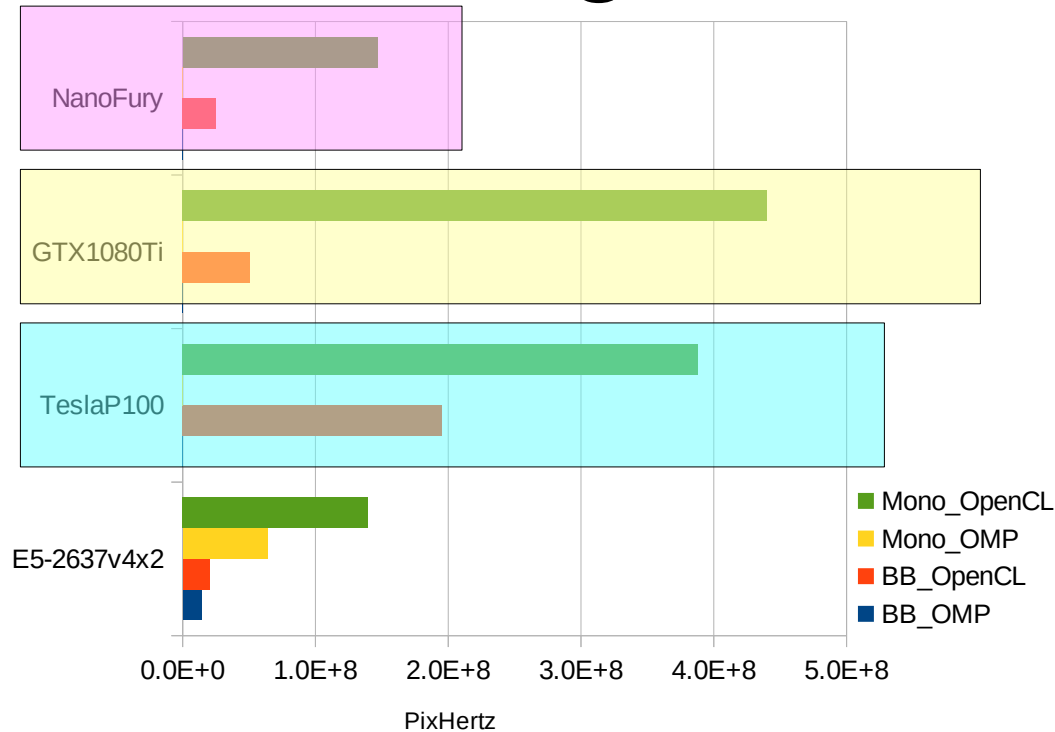
En 2013, arrivée de Kepler et Phi

Distinction en GPU & GPGPU



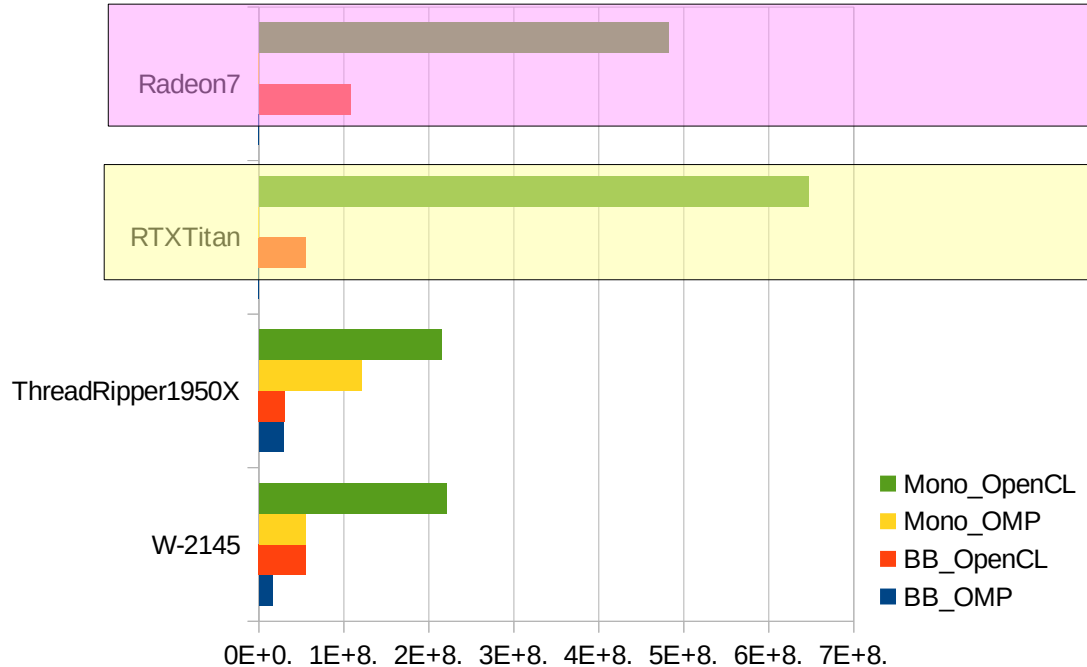
- Un x4 en Mono et en BB pour le GPU GTX780Ti
- Un x10 en BB et seulement x3 en Mono pour le GPGPU K40m
- Un x4 en BB et seulement un x1.4 en Mono pour le MIC Phi 7120P

En 2016, Pascal, Broadwell, Fiji 3 nouvelles générations



- La Gamer l'emporte sur la GPGPU en Mono (légèrement)
- La Tesla met un x4 au GPU, un x8 à AMD et un x10 au CPU
- Le AMD Fiji est au niveau du CPU Broadwell

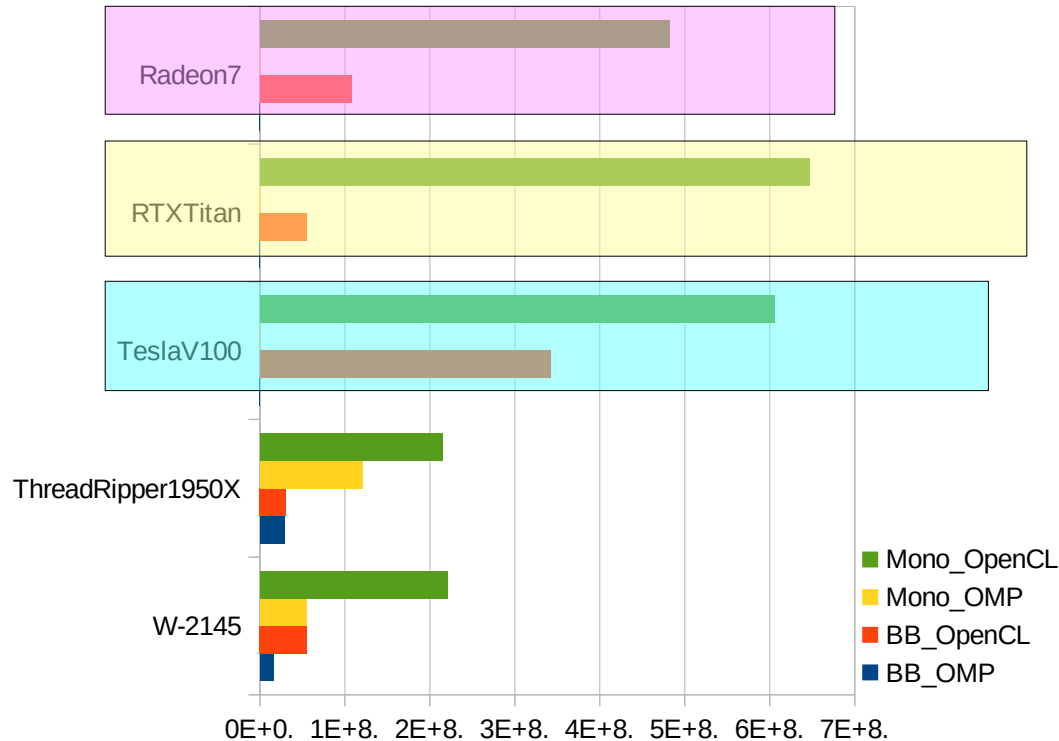
En 2019, Turing, Vega2 en GPU Skylake, Zen en CPU, **sans** V100



- Un Zen 16 cœurs d'AMD ~ un Skylake 8 cœurs Intel
- Radeon 7 : x2.5 en Mono, x2 en BB (vs Best CPU)
- Turing RTX : x3 en Mono mais comparable en BB (vs Best CPU)

Sortie en 2017, la Tesla V100

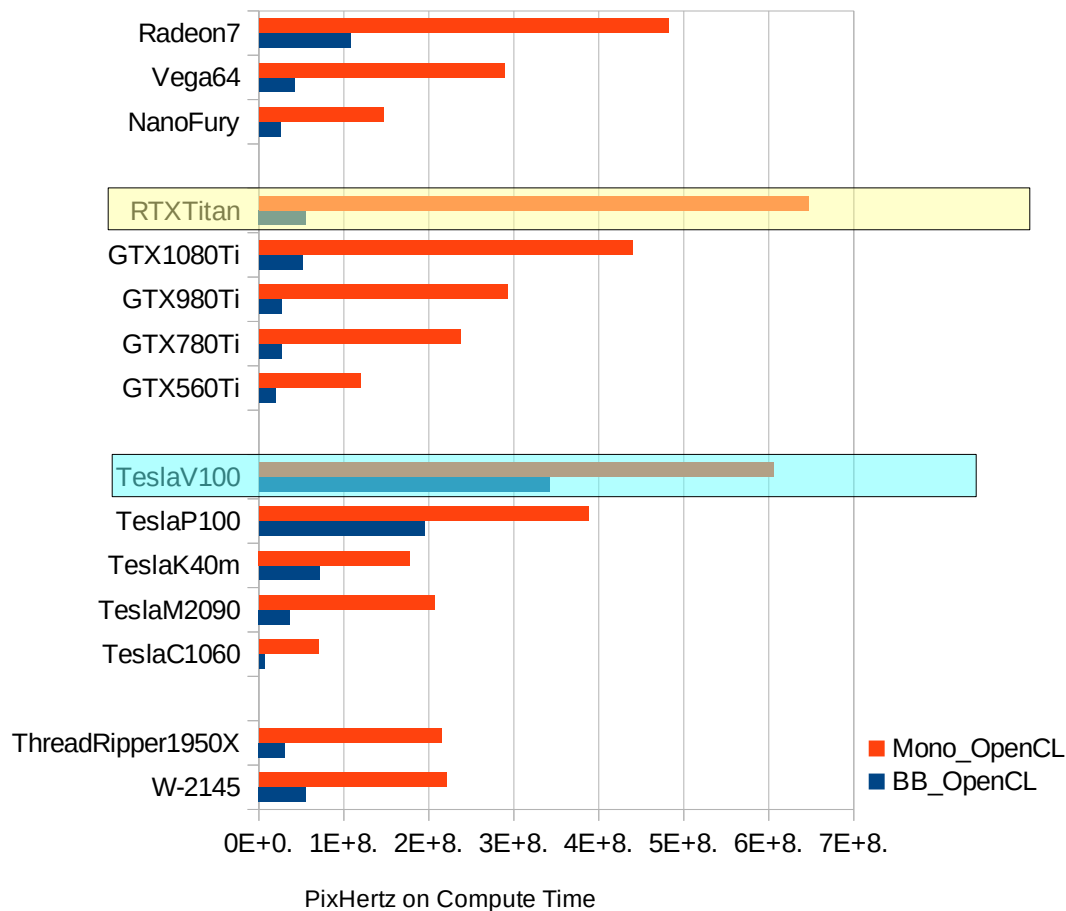
Seulement fin 2019 au CBP



- Tesla V100 ~ RTX Titan en Mono, x6 en BB
- Tesla V100 ~ Radeon 7 en Mono, x3 en BB
- CPU à x3 en Mono, mais x10 en BB

Synthèse en OpenCL

GPU vs meilleurs des CPU



Selon la simulation (BB ou Mono), c'est Gamer ou Tesla !

Et CUDA dans tout ça ?

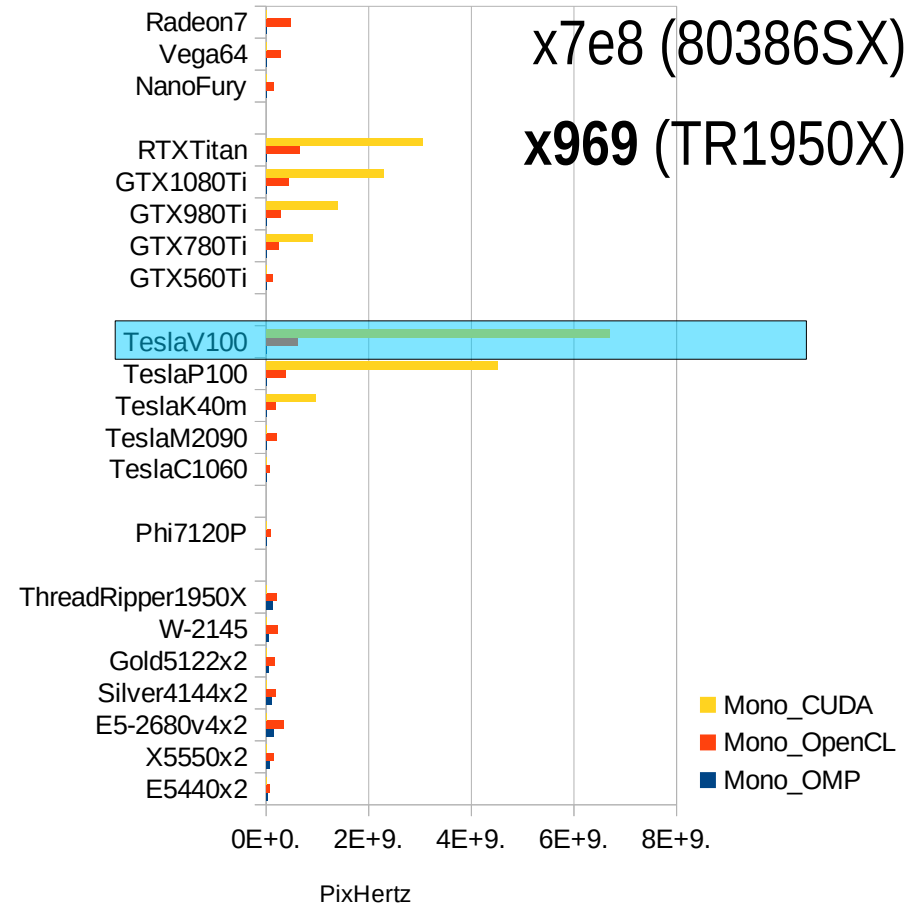
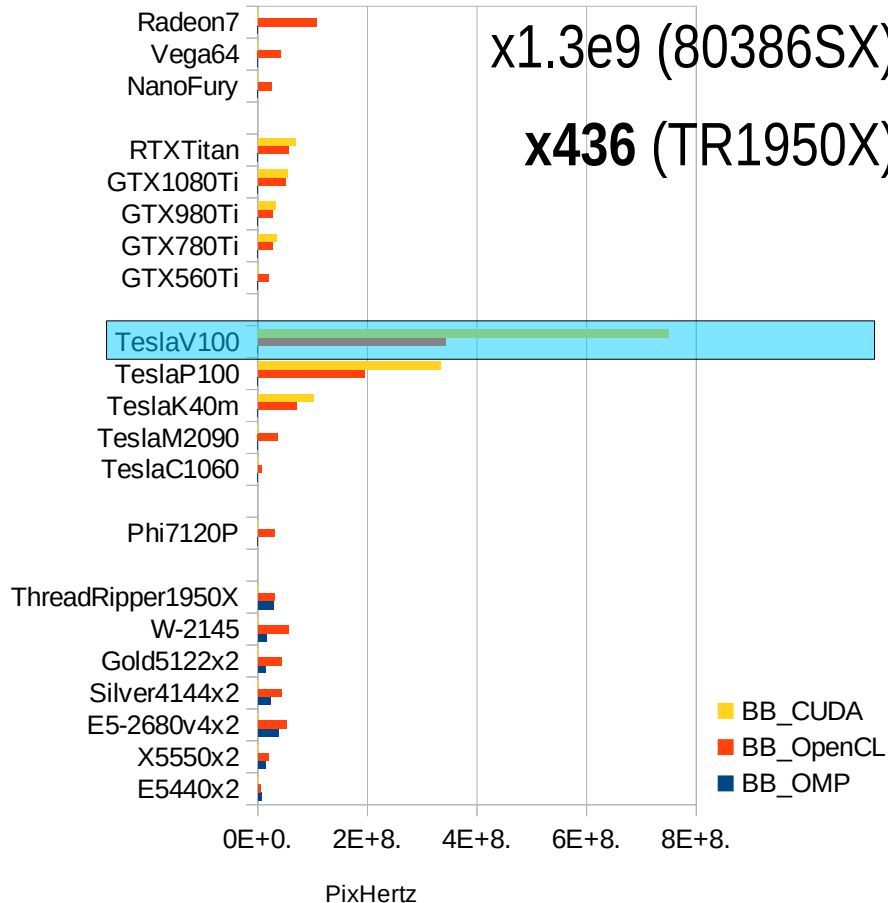
d'OpenCL en CUDA avec Python

- Approche « naïve » :
 - Copier/Coller de tous les noyaux OpenCL
 - Remplacement des « workitem » par les « BlockIdx »
 - Préfixe des fonctions « internes » des noyaux par « __device__ »
- Petit test en BB, Tesla P100 :
 - En OpenCL : 1.031377 s
 - En CUDA : 24.033100 s (**soit 24 fois plus lent !!!**)
- Petit test en Mono, Tesla P100 :
 - En OpenCL : 0.87633 s
 - En CUDA : 1.003969 s (**un peu moins bon**)
- Bref, ça rappelle un souci déjà rencontré...

En CUDA, l'attaque des threads...

- Exploitation de 2 étages de parallélisme
 - Définition de « grid » et de « threads »
 - Découpage dans les noyaux des tâches mode « pagination »
 - En OpenCL : `uint xi=(uint)get_global_id(0);`
 - En CUDA : `uint xi=(uint)(blockIdx.x*blockDim.x+threadIdx.x);`
- Nouveau test en BB, Tesla P100, Mode TrajectoPixel :
 - Threads à 1 : 24.033100 s
 - Threads à 32 et 32^2 : 0.810377 s (soit x30/CUDA₁, légèrement meilleur OpenCL)
- Nouveau test en Mono, Tesla P100, Mode TrajectoPixel :
 - Threads à 1 : 1.003969 s
 - Threads à 32 et 32^2 : 0.063761 (soit x15/CUDA, x13/OpenCL)
- En passant en CUDA ; +20 % en BB mais x13 en Mono !

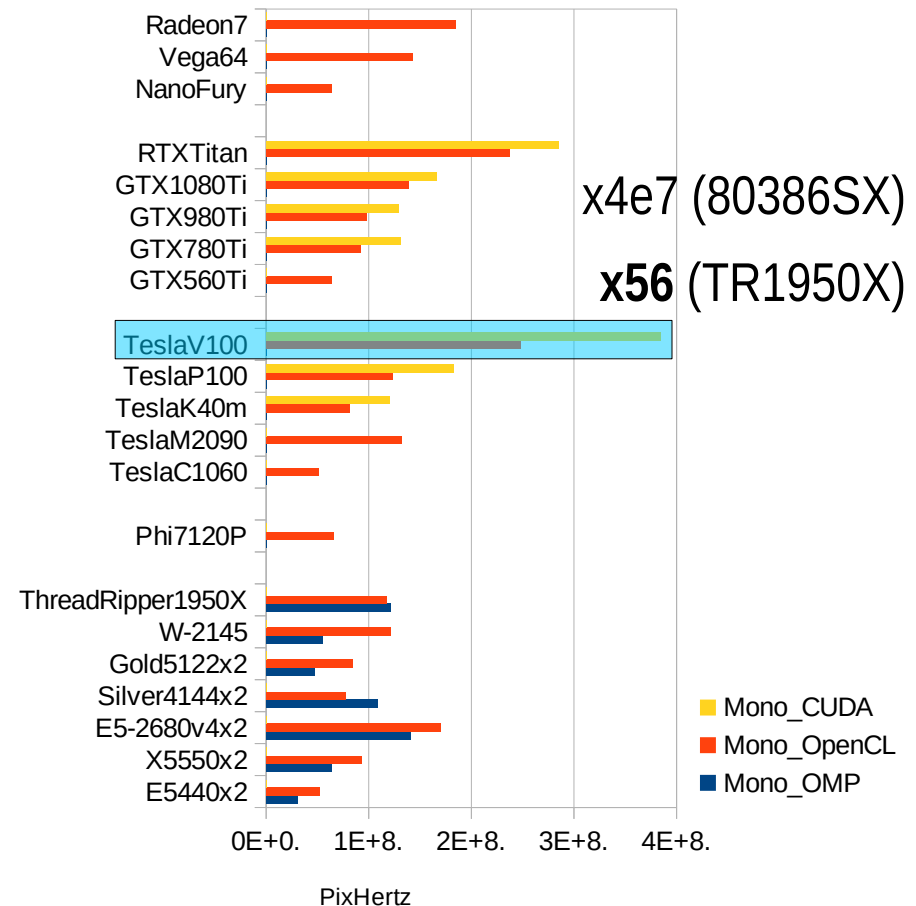
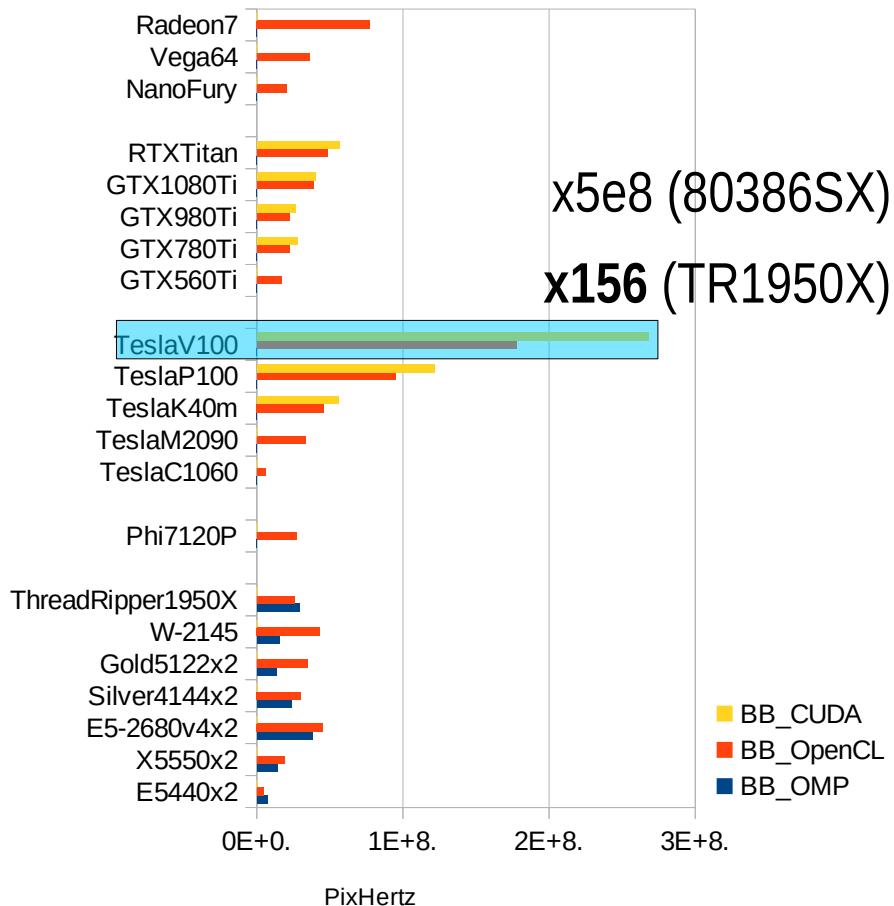
Donc, en rajoutant CUDA, Il y a la Tesla V100 et les autres...



Face au CPU, le milliard en 30 ans, le millier en séquentiel

Mais je serai un gros menteur...

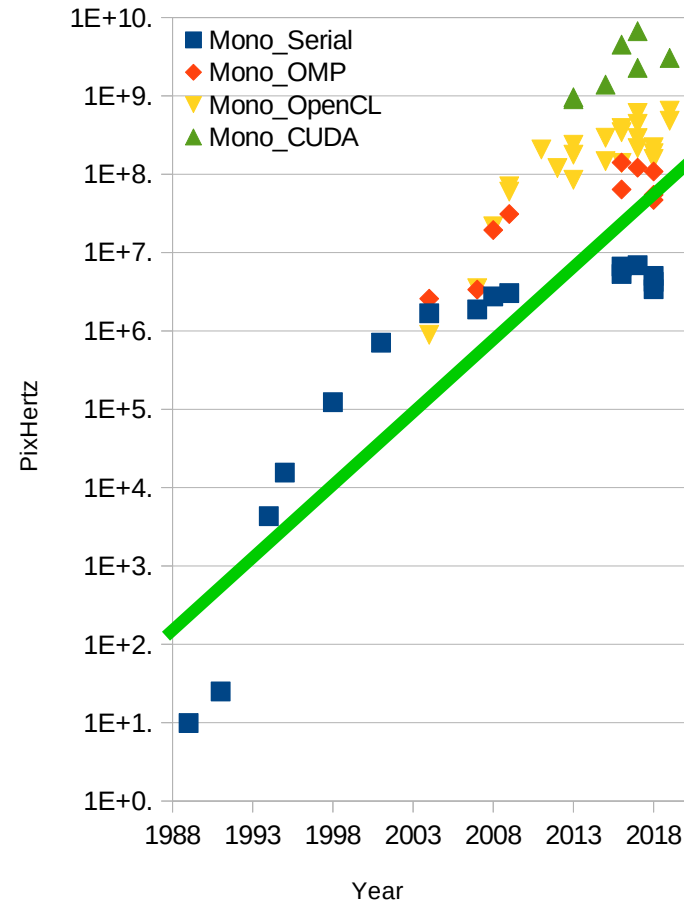
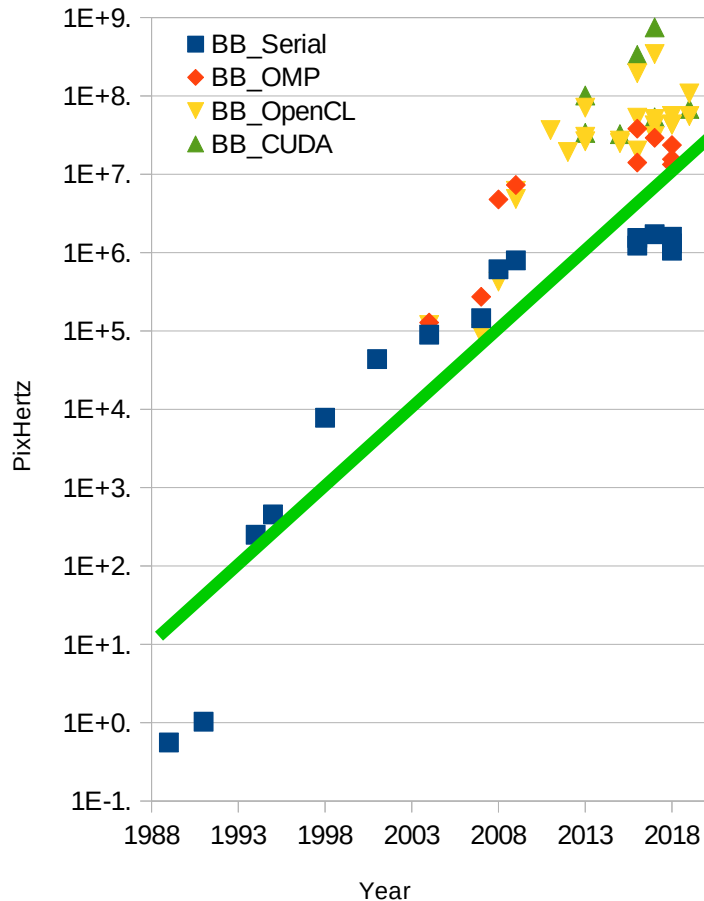
Le transfert en CUDA et OpenCL...



Bref, dans ce cas, temps du transfert > temps de calcul

Et la loi de Moore* dans tout ça ?

Grâce à CUDA, on est sur la droite !



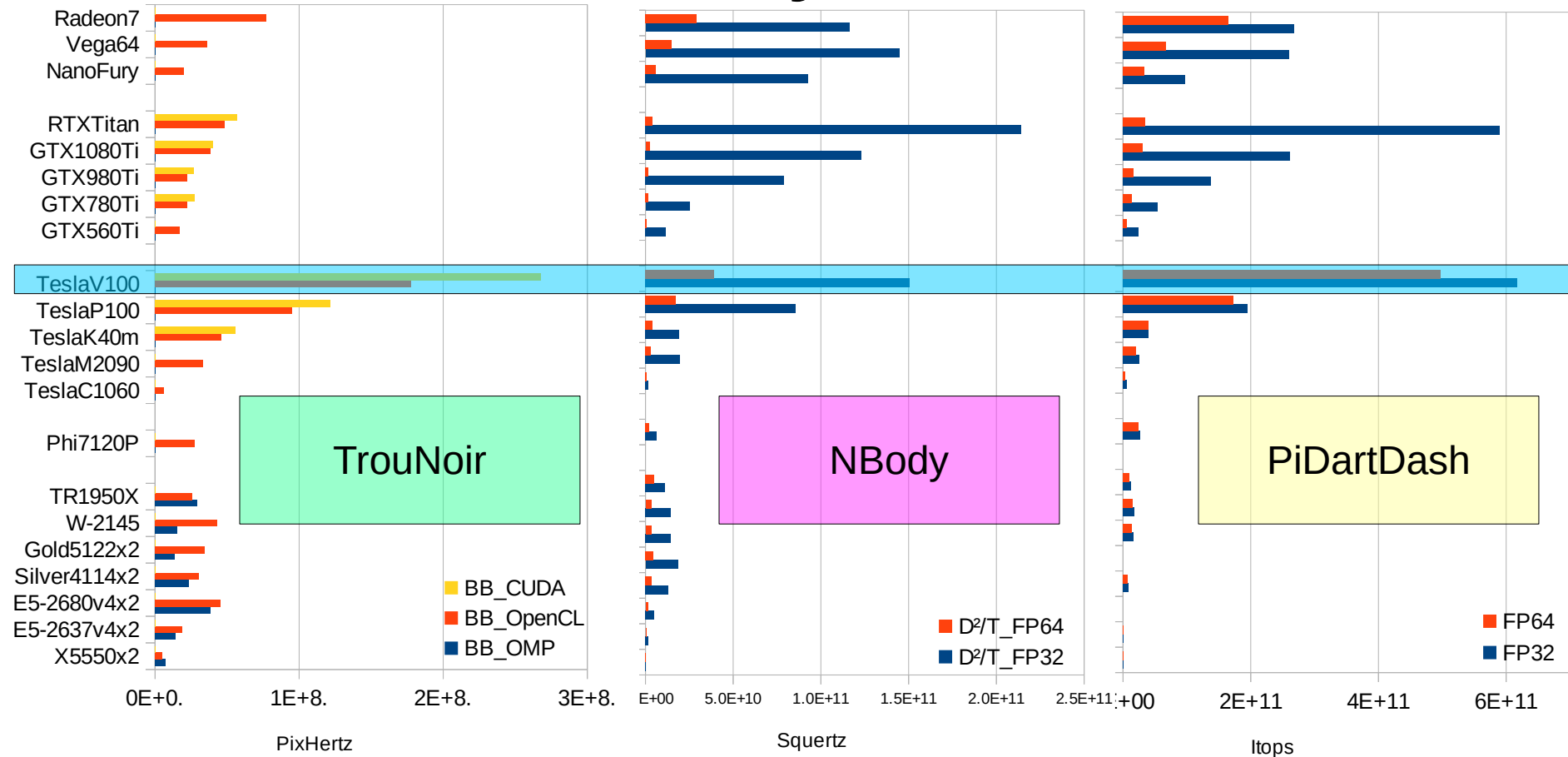
Le (GP)GPU : même caractère « disruptif » que le FPU...

En conclusion de ce portage... ... et son exécution sur nos *PU

- Le Code, le chemin & les gains (temps écoulés) :
 - De Fortran à C en 1997 : x1
 - De C à C/OMP en 2019 : x22 (pour un 28 cœurs)
 - De C/OMP à Python/OpenCL sur CPU : x25 (pour un 8 cœurs)
 - Du CPU au GPU en Python/OpenCL : de x36 à x106
 - De Python/OpenCL à Python/CUDA : de x56 à x156
- Bref...
 - Il n'y a pas que CUDA ou C++ dans la vie... Il y a Python et OpenCL !
 - Vous pouvez tester :
 - <http://forge.cbp.ens-lyon.fr/redmine/projects/bench4gpu/repository/show/TrouNoir>

Comparaison aux autres codes...

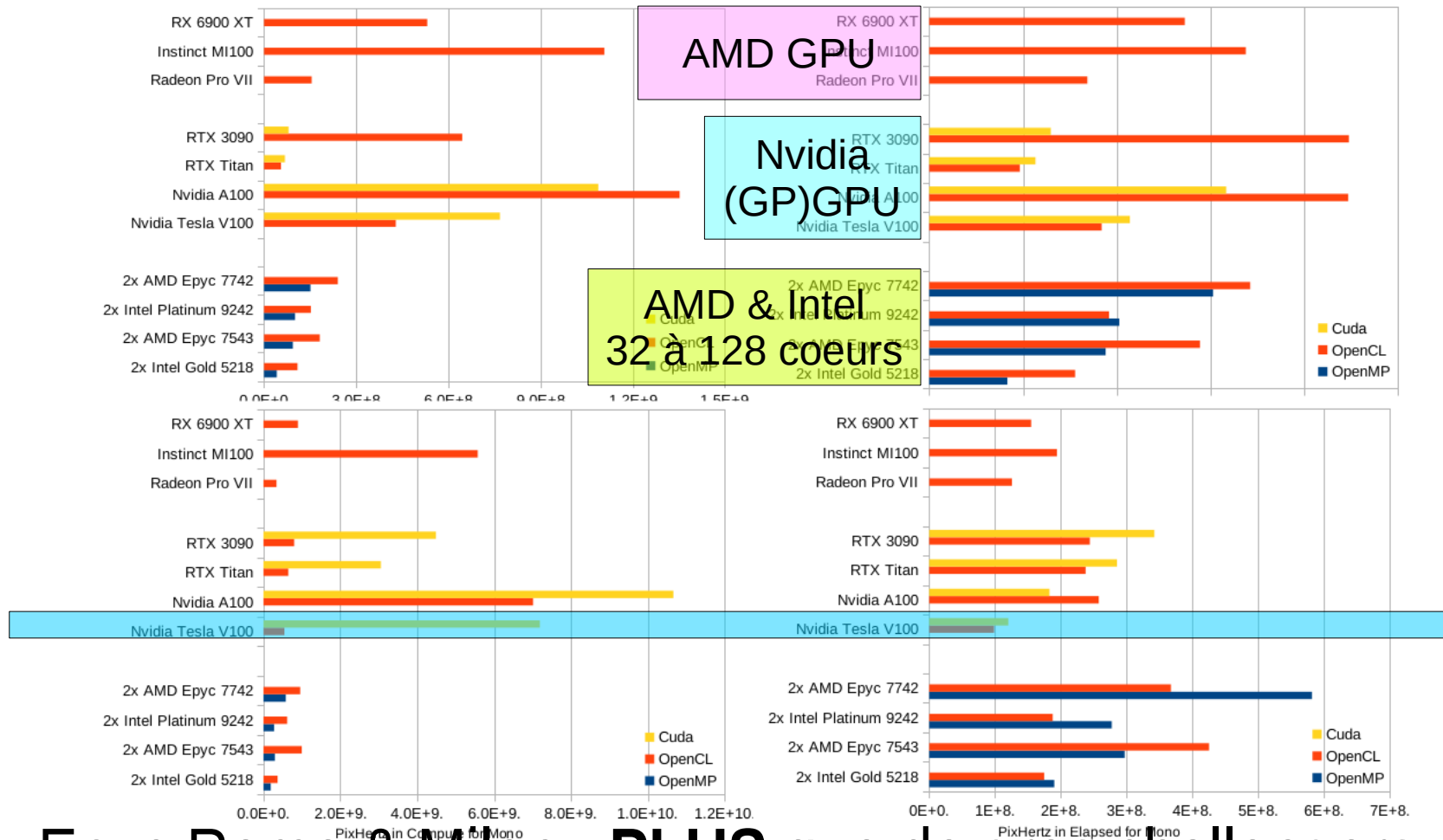
TrouNoir, Nbody et PiDartDash



En conclusion, pour les GPU, calculs pas trop compliqués...

2 ans plus tard, larges évolutions !

Ampere, Navi, Rome, Cascade



Epyc Rome & Milan : **PLUS** que de gros challengers...

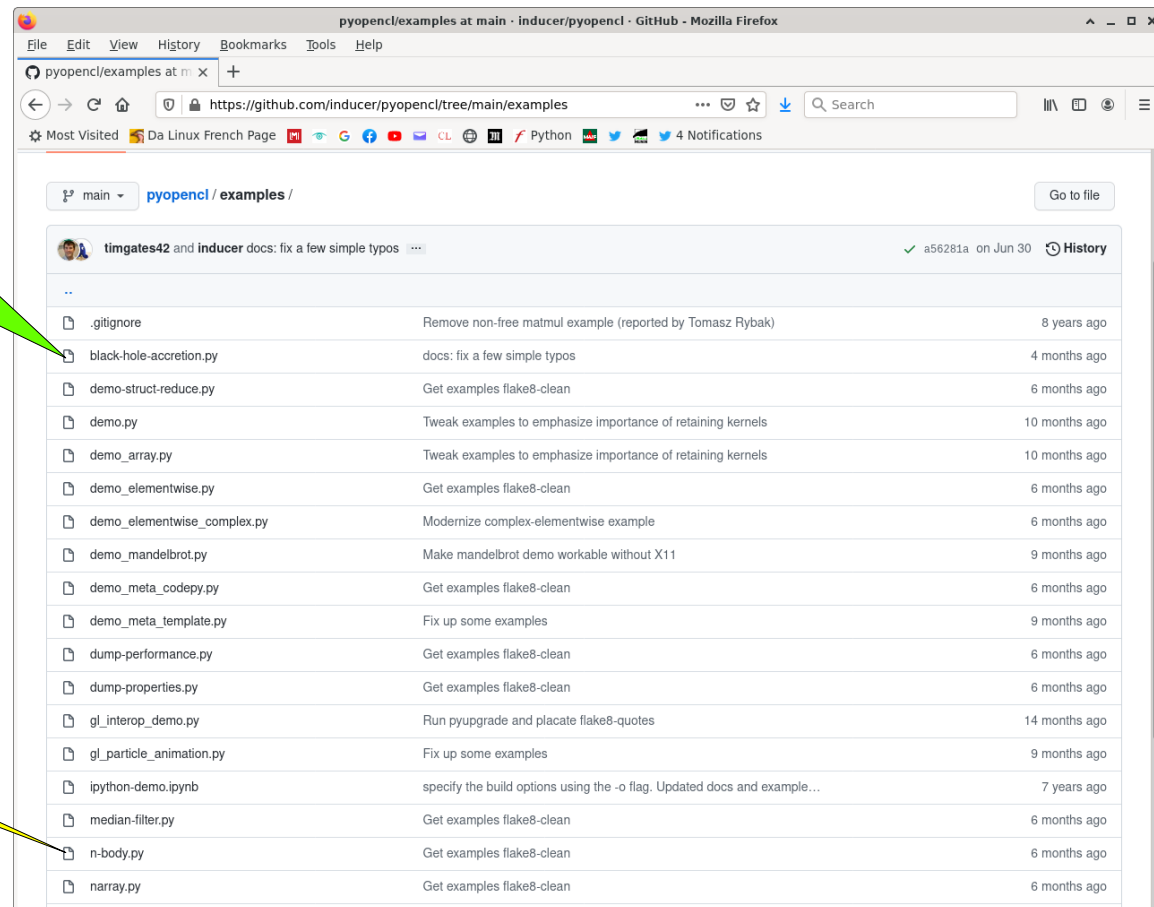
Et « TrouNoir.py » maintenant...

Devenu « black-hole-accretion.py »

Site PyOpenCL : <https://github.com/inducer/pyopencl/tree/main/examples>

black-hole-accretion.py

n-body.py



pyopencl/examples at main · inducer/pyopencl · GitHub - Mozilla Firefox		
pyopencl/examples at main · inducer/pyopencl · GitHub - Mozilla Firefox		
https://github.com/inducer/pyopencl/tree/main/examples		
Most Visited Da Linux French Page Python 4 Notifications		
main pyopencl / examples /		
timgates42 and inducer docs: fix a few simple typos a56281a on Jun 30 History		
..		
.gitignore	Remove non-free matmul example (reported by Tomasz Rybak)	8 years ago
black-hole-accretion.py	docs: fix a few simple typos	4 months ago
demo-struct-reduce.py	Get examples flake8-clean	6 months ago
demo.py	Tweak examples to emphasize importance of retaining kernels	10 months ago
demo_array.py	Tweak examples to emphasize importance of retaining kernels	10 months ago
demo_elementwise.py	Get examples flake8-clean	6 months ago
demo_elementwise_complex.py	Modernize complex-elementwise example	6 months ago
demo_mandelbrot.py	Make mandelbrot demo workable without X11	9 months ago
demo_meta_codepy.py	Get examples flake8-clean	6 months ago
demo_meta_template.py	Fix up some examples	9 months ago
dump-performance.py	Get examples flake8-clean	6 months ago
dump-properties.py	Get examples flake8-clean	6 months ago
gl_interop_demo.py	Run pyupgrade and placate flake8-quotes	14 months ago
gl_particle_animation.py	Fix up some examples	9 months ago
ipython-demo.ipynb	specify the build options using the -o flag. Updated docs and example...	7 years ago
median-filter.py	Get examples flake8-clean	6 months ago
n-body.py	Get examples flake8-clean	6 months ago
narray.py	Get examples flake8-clean	6 months ago

En conclusion pour TrouNoir.py...

- L'informatique a évolué en 30 : des gains colossaux
 - Principaux gains : FPU (début 80), GPU (début 2010)...
- En 2021, puissance de calcul : GPGPU puis HugeCPU
- Paralléliser ou gépuifier un code : une question de temps
 - C'est possible, mais il faut qu'il soit bien « conditionné »
 - En OpenMP, parfois, distribuer des boucles suffit
 - En OpenCL, copier son code C dans du Python et appeler les noyaux
 - En CUDA, s'inspirer de OpenCL et exploiter les « étages » de //isation
- Constater que tout n'est que « cas d'usage »

Vers un portage préliminaire...

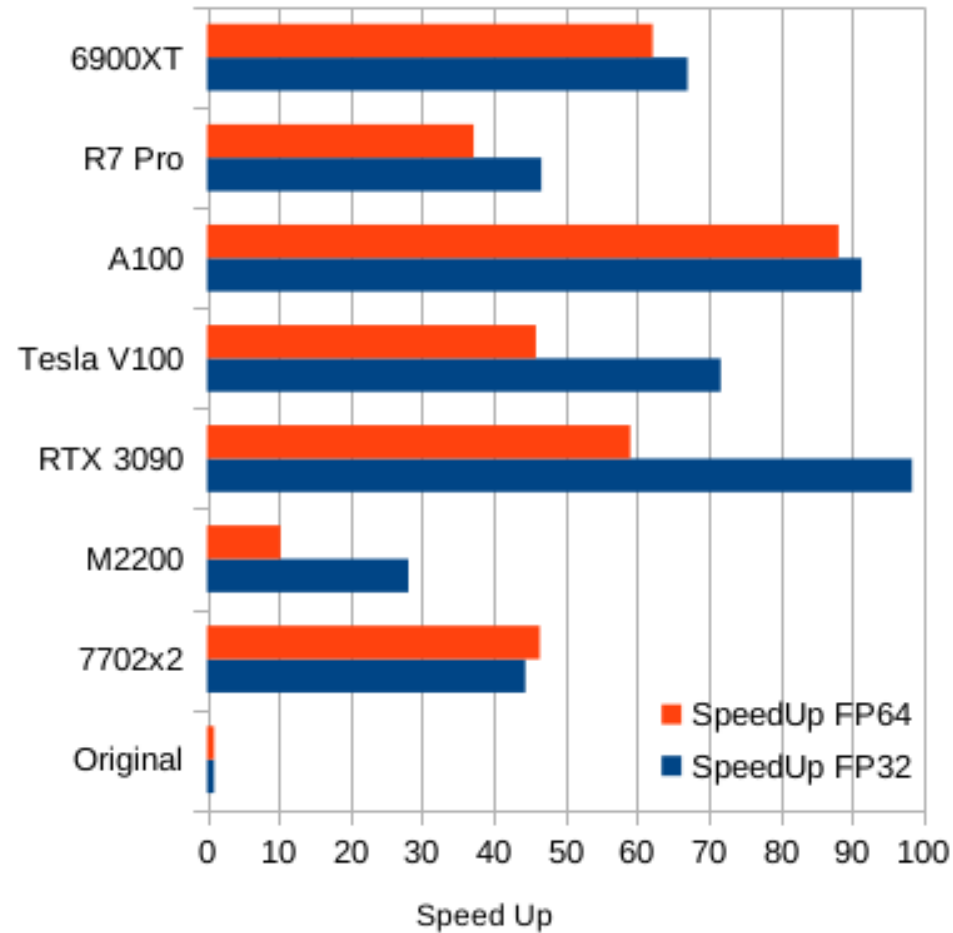
Application SPH en astrophysique

- Contexte de l'étude :
 - Code issu d'un développement formel sous Mathematica
 - En C++, 2 millisecondes par particule
 - Nécessité de lancer le lancer sur un million de particules
- Objectif : réalisation d'un démonstrateur
 - Portage en Python/OpenCL
 - Comparaison de fichiers de sortie
 - Premier portage : utilisation de « TrouNoir.py » comme « matrice »

Premiers résultats

Encourageants mais déstabilisants

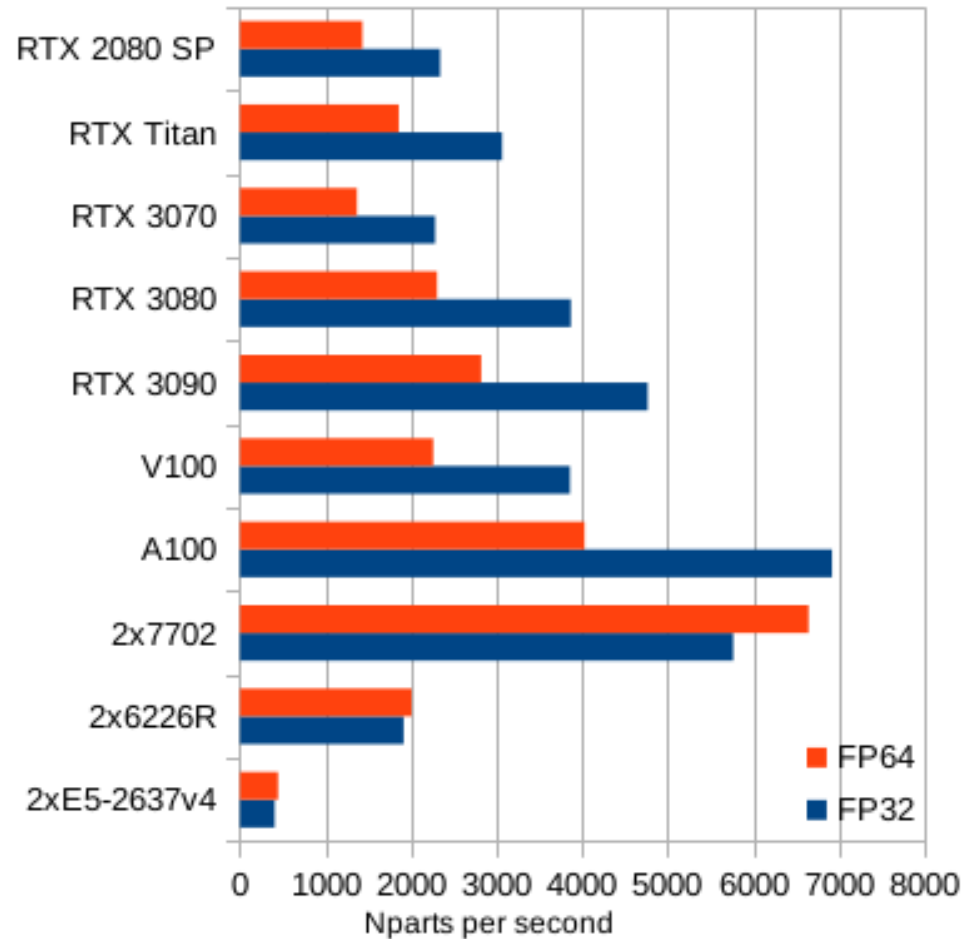
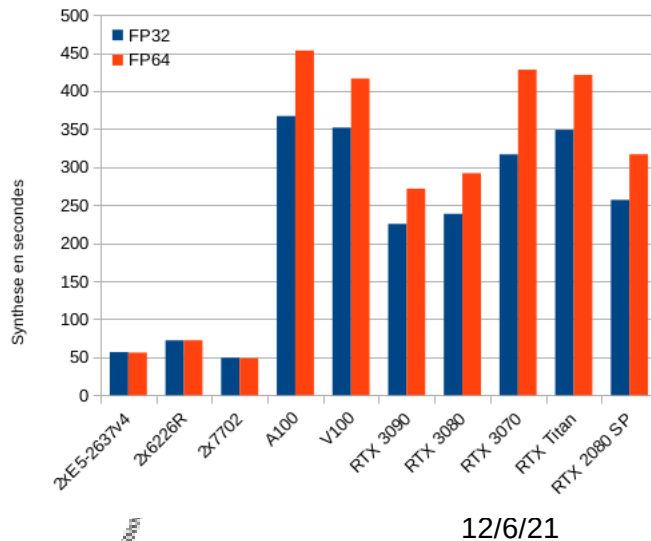
- « Noyaux » OpenCL :
 - du copier/coller
- Essentiel du temps passé :
 - formatage sortie C++...
- Speed Up « confortable »
 - ~45 pour les meilleurs CPU
 - ~90 pour le meilleur GPU
- Proximité FP32 & FP64
 - Code « Memory Bound »



Ordre #1 : des noyaux « énormes »

Les « HugeCPU » impressionnants

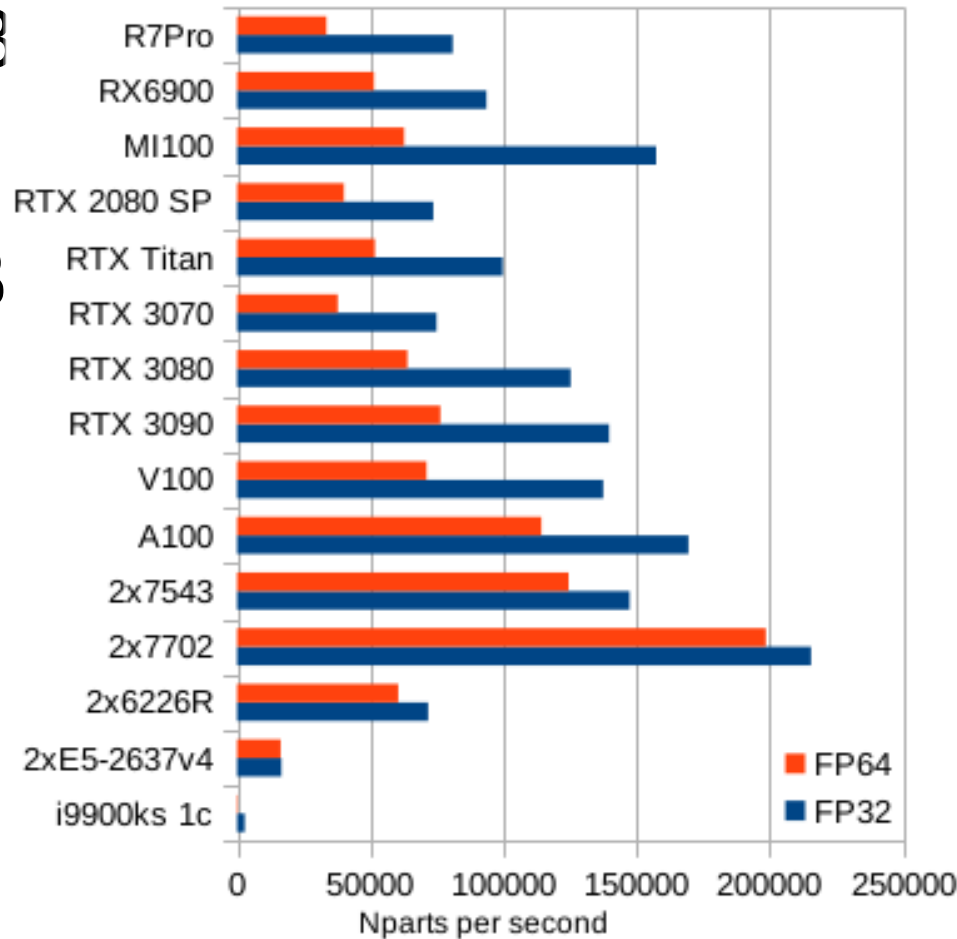
- Ordre #0 : « wc », 26KB
 - 860 lignes, 2057 mots
- Ordre #1 : « wc », 28MB
 - 6581 lignes, 31191 mots
- « Synthèse » préalable...



Ordre #2 : des noyaux « purgés »

Les « HugeCPU » l'emportent...

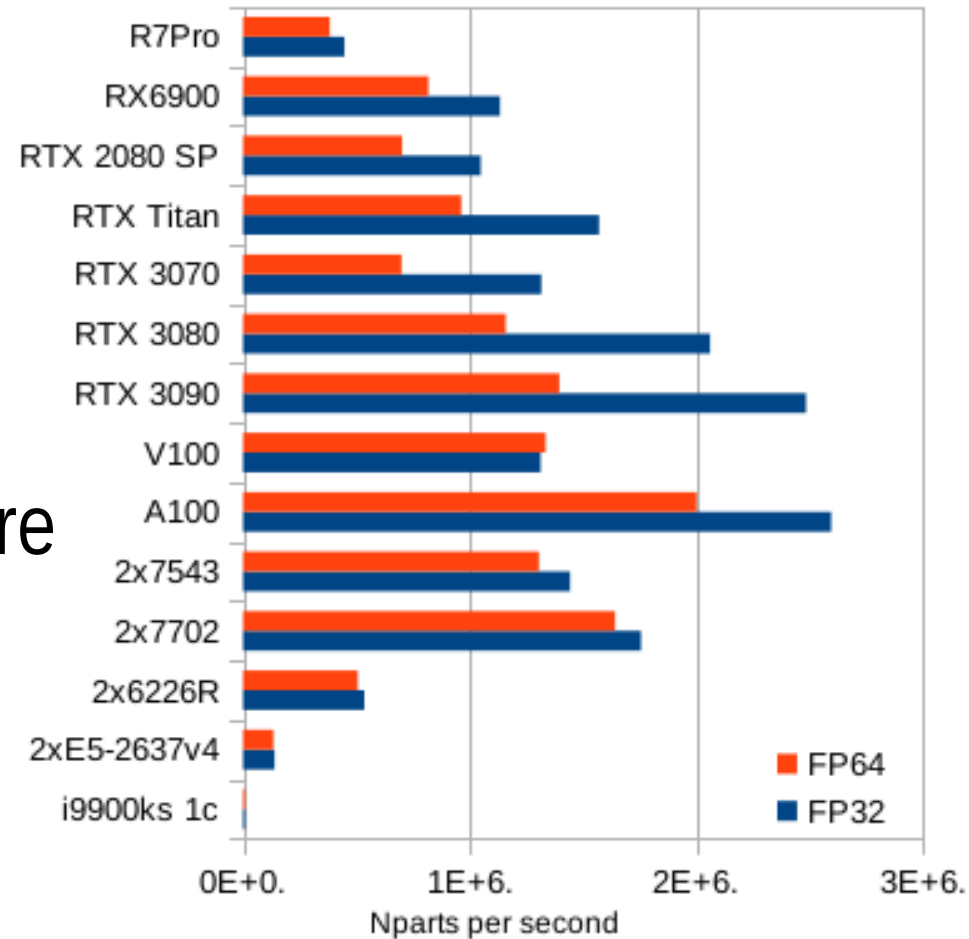
- Ordre #1 : « wc » de 372KB
 - 5020 lignes, 15423 mots
- Ordre #2 : « wc » de 3.7MB
 - 6132 lignes, 20054 mots
- « Synthèse » ~ seconde
- Résultats
 - HugeCPU en tête
 - HugeGPGPU sur le podium



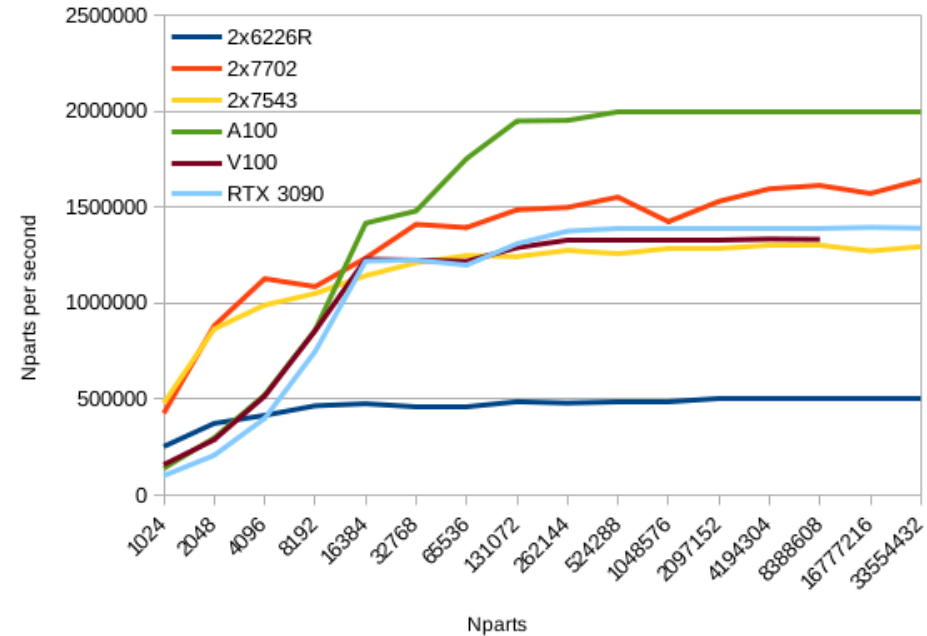
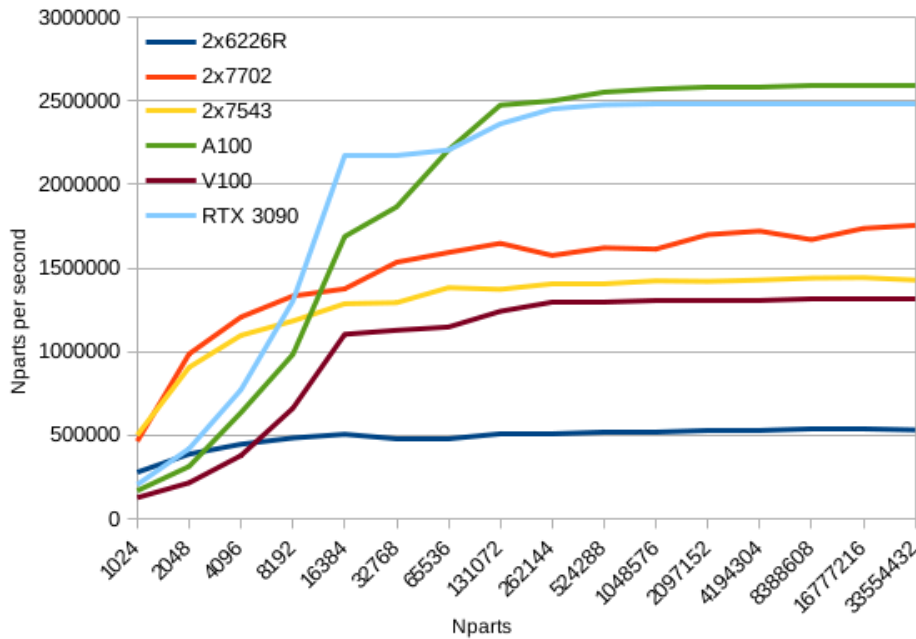
Ordre #2 : nouvelle implémentation

Les (GP)GPU d'une courte tête...

- Un gain de 10 au total
 - L'algorithme a du bon
- La Nvidia A100 en tête
 - Les Ampère Gamer derrière
- Les HugaGPU juste derrière
 - Milan bien meilleur que Rome
- Confirmation...



Conclusion du démonstrateur « Montée en charge » : le critère...

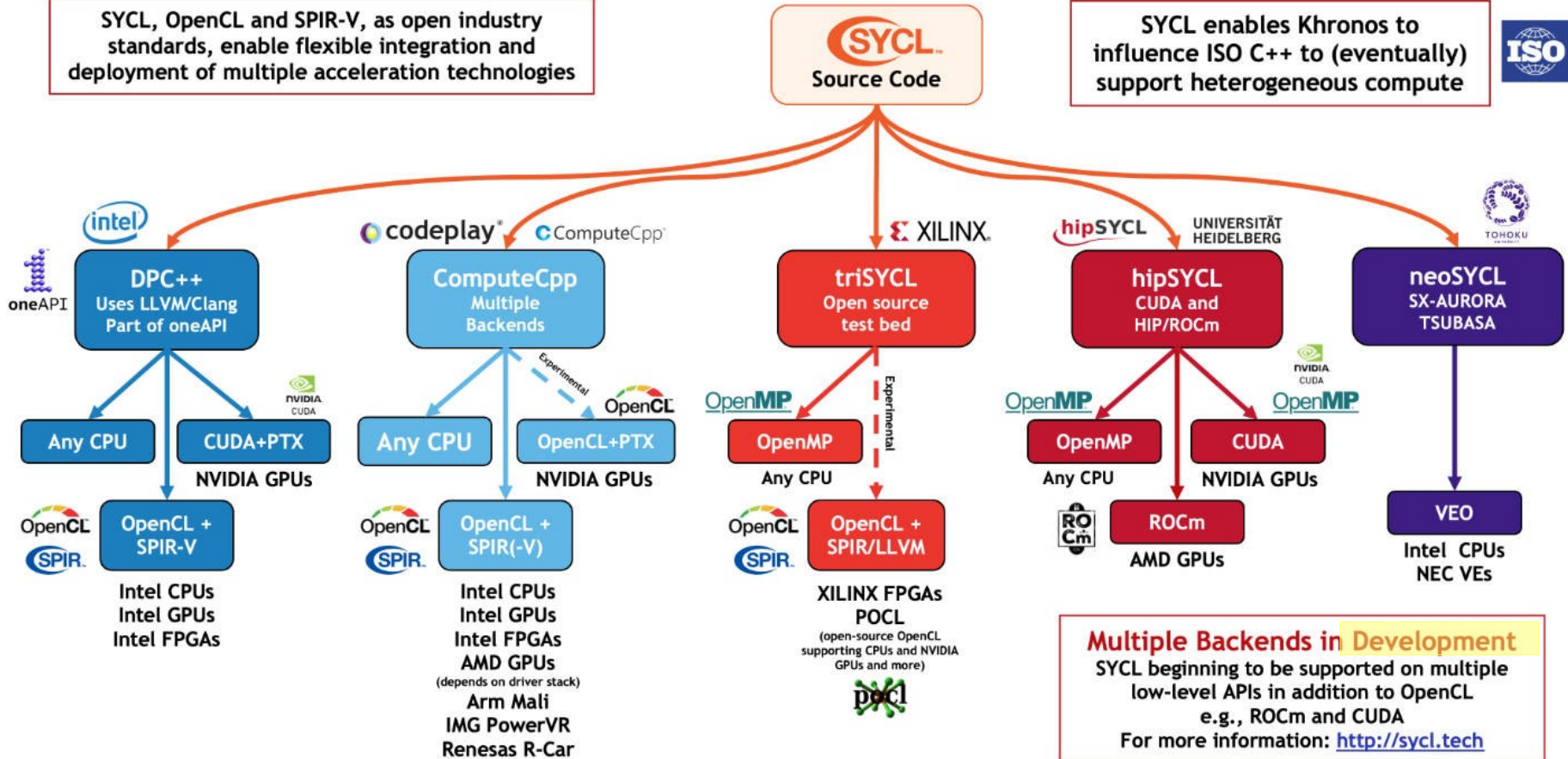


- Distribution optimale : > à 200000...
- Equipe CRAL : préférence pour SyCL
 - Un doctorant « démerdard » comme référence interne...

SyCL : « l'avenir » de OpenCL ?

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute



- En fait, TOUT est en développement...

SyCL Intel sur GitHub : Juste pour donner une idée...

Build DPC++ toolchain with support for NVIDIA CUDA

There is experimental support for DPC++ for CUDA devices.

To enable support for CUDA devices, follow the instructions for the Linux or Windows DPC++ toolchain, but add the `--cuda` flag to `configure.py`. Note, the CUDA backend has experimental Windows support, windows subsystem for linux (WSL) is not needed to build and run the CUDA backend.

Enabling this flag requires an installation of [CUDA 10.2](#) on the system, refer to [NVIDIA CUDA Installation Guide for Linux](#) or [NVIDIA CUDA Installation Guide for Windows](#)

Currently, the only combination tested is Ubuntu 18.04 with CUDA 10.2 using a Titan RTX GPU (SM 71). The CUDA backend should work on Windows or Linux operating systems with any GPU compatible with SM 50 or above. The default SM for the NVIDIA CUDA backend is 5.0. Users can specify lower values, but some features may not be supported. Windows CUDA support is experimental as it is not currently tested on the CI.

Non-standard CUDA location

If the CUDA toolkit is installed in a non-default location on your system, two considerations must be made.

Firstly, **do not** add the toolkit to your standard environment variables (`PATH` , `LD_LIBRARY_PATH`), as to do so will create conflicts with OpenCL headers.

Secondly, set the `CUDA_LIB_PATH` environment variable and pass the CMake variable `CUDA_TOOLKIT_ROOT_DIR` as follows:

```
CUDA_LIB_PATH=/path/to/cuda/toolkit/lib64/stubs CC=gcc CXX=g++ python $DPCPP_HOME/llvm/buildbot/configure.py --cuda --cmake
CUDA_LIB_PATH=/path/to/cuda/toolkit/lib64/stubs CC=gcc CXX=g++ python $DPCPP_HOME/llvm/buildbot/compile.py
$DPCPP_HOME/llvm/build/bin/clang++ -std=c++17 -O3 -fsycl -fsycl-targets=nvptx64-nvidia-cuda --cuda-path=/path/to/cuda/toolkit
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$DPCPP_HOME/llvm/build/lib ./a.out
```

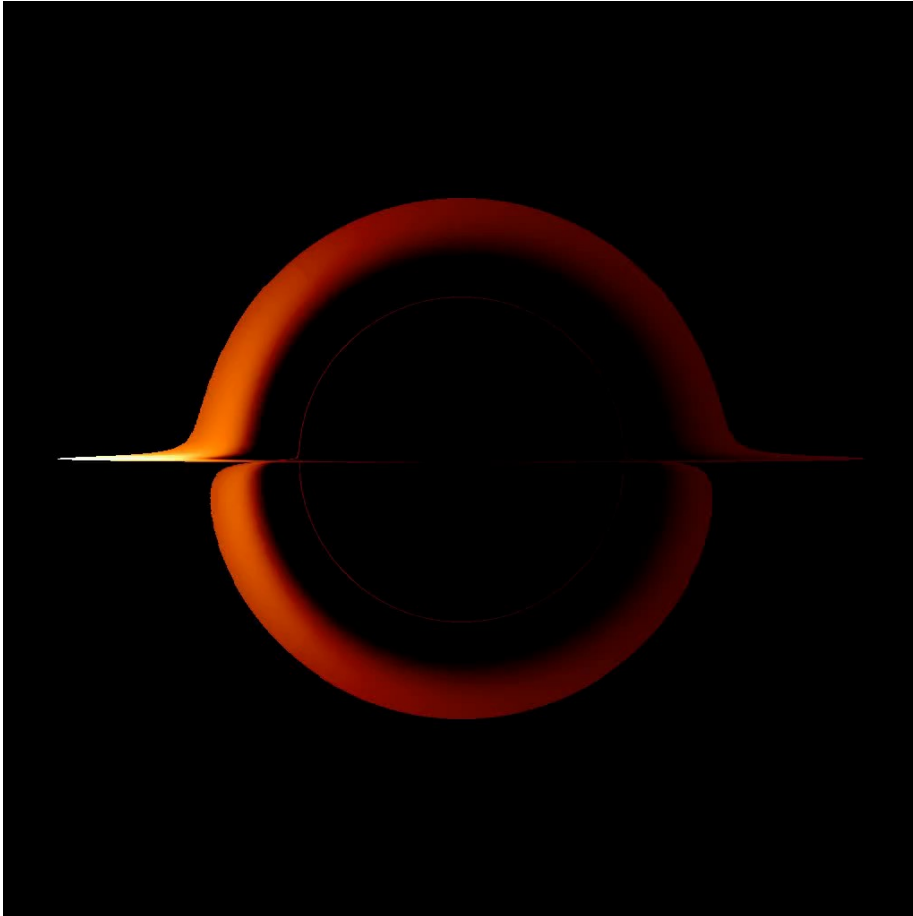
- Ne fonctionne « que » sur les pilotes Nvidia « externes »

Annuaire de « bonnes pratiques »

Pour éviter la déception...

- Le constat de 2021 (différent de 2019) :
 - Le (GP)GPU est performant, mais plus le meilleur
- Se poser les « bonnes questions » :
 - Combien de tâches indépendantes puis-je lancer ? >100000 ?
 - Quelles sont les communications entre tâches ?
 - Quel est le « grain » de mon application ?
 - Quelle doit être la pérennité de mon code ?
 - Quel doit être l'oecuménisme de mon code ?

Des questions ?



- Remerciements à :
 - Pierre Léna
 - Jean-Pierre Luminet
 - Didier Pelat
 - Hervé Aussel
 - Ralf Everaers
 - Hervé Gilquin
 - Simon Delamare
 - Marine, Yoann, Léna