4D Genome Computational Day

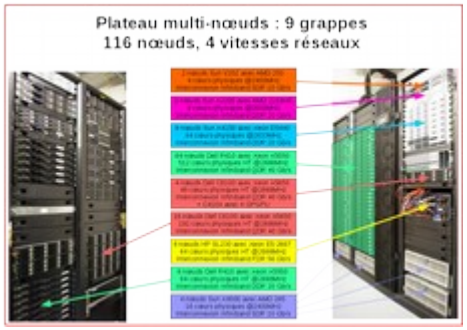
Exploit GPUs from an experimental point of view : program from scratch or migrate old code ?

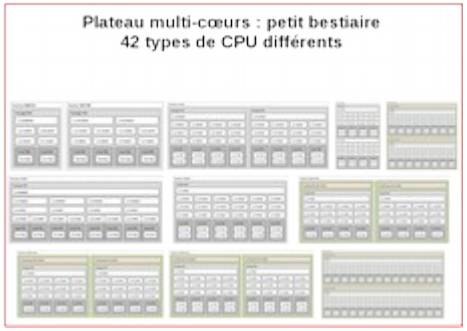
Emergence of “Matrix codes” ?

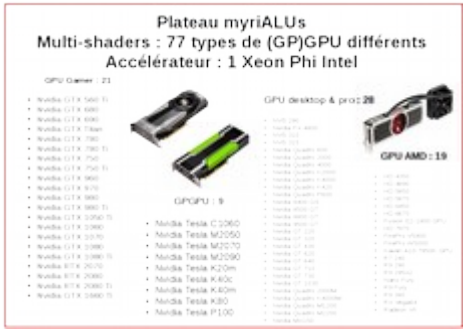
Emmanuel Quémener

Blaise Pascal Center And its Test Center...

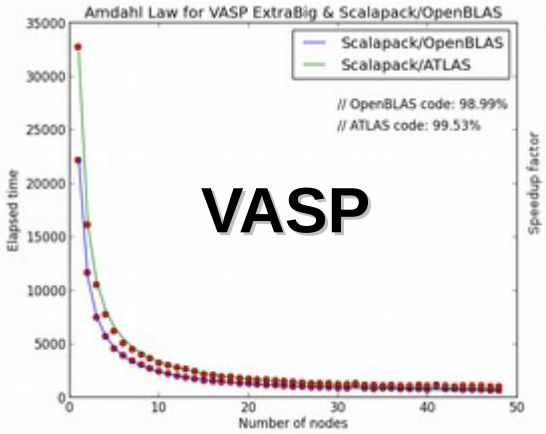
- Centre Blaise Pascal : 3 hostings
  - Hotel for conferences
  - Hotel for formations
  - Hotel for projects
- Test center : 3 quests
  - Reproducibility
  - Scalability
  - Simplicity
- Its own technical benchs

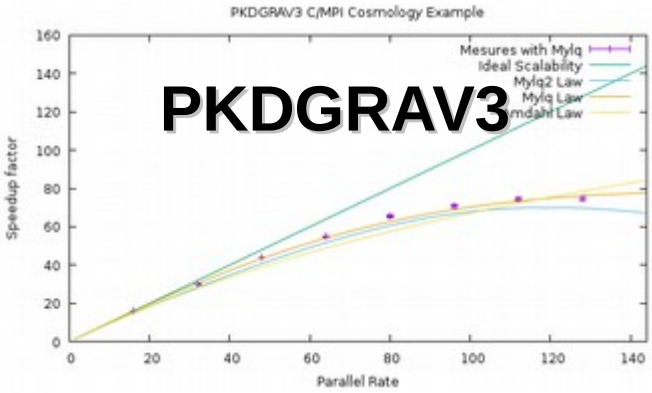


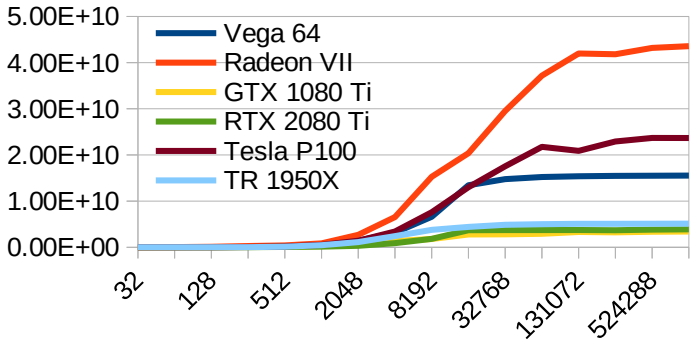


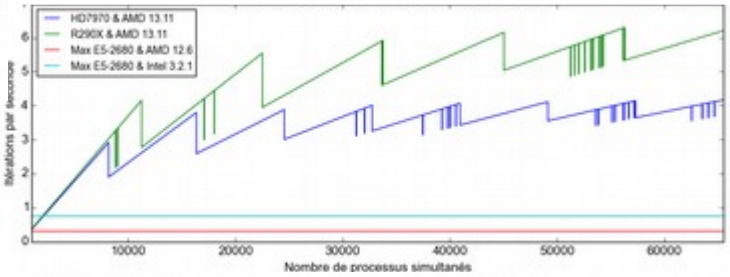


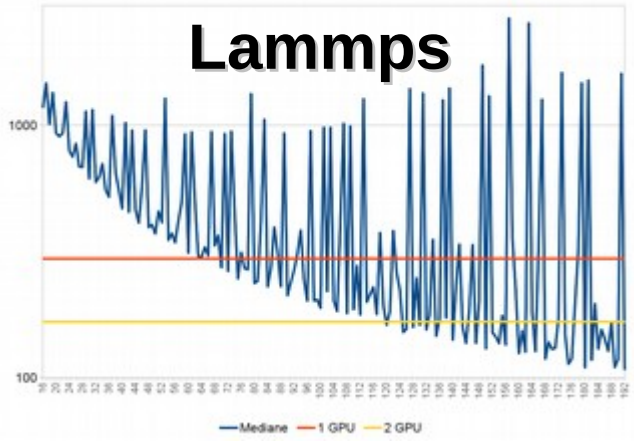
« Business » codes & hardware
First steps in scalability

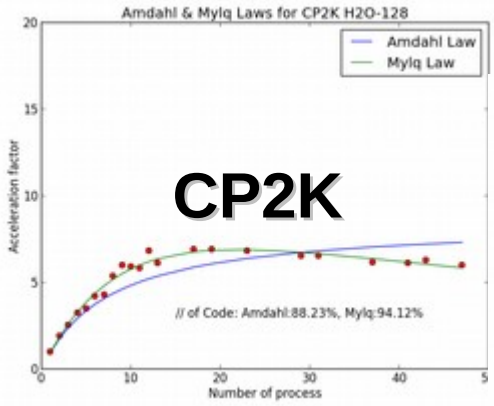


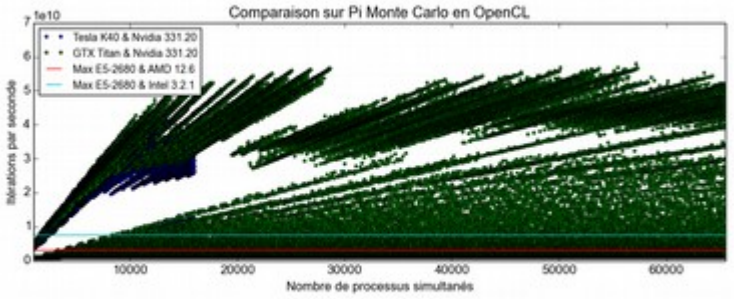












Multinodes TechBench : 9 clusters 116 nodes, 4 IB speeds

2 nodes Sun V20Z with AMD 250
4 physical cores @2400MHz
Interconnection Infiniband SDR 10 Gb/s

2 nodes Sun X2200 with AMD 2218HE
8 physical cores @2600MHz
Interconnection Infiniband DDR 20 Gb/s

8 nodes Sun X4150 with Xeon E5440
64 physical cores @2833MHz
Interconnection Infiniband DDR 20 Gb/s

64 nodes Dell R410 with Xeon X5550
512 physical cores HT @2666MHz
Interconnection Infiniband QDR 40 Gb/s

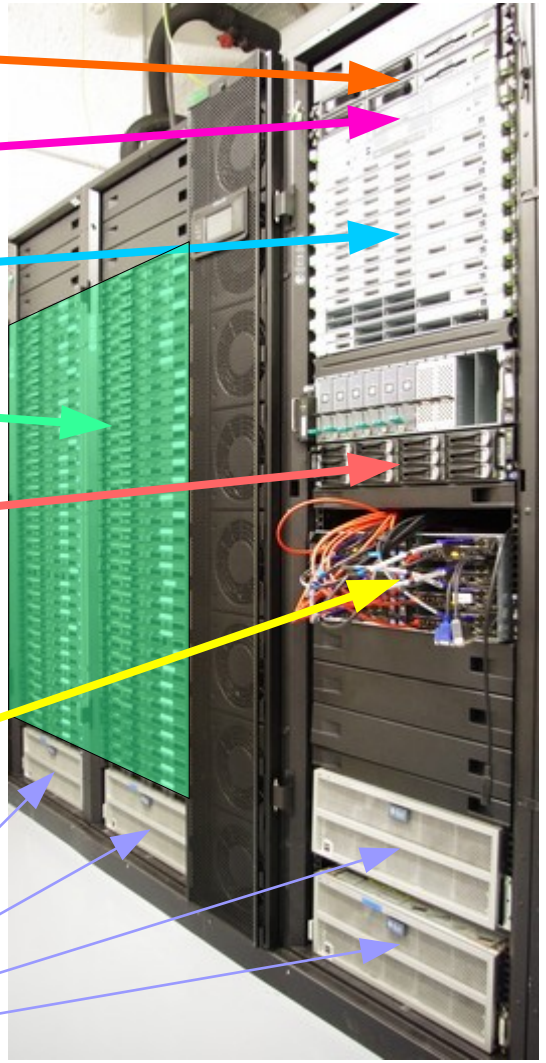
4 nodes Dell C6100 with Xeon X5650
48 physical cores HT @2666MHz
Interconnection Infiniband QDR 40 Gb/s
+ C410X with 4 GPGPU

16 nodes Dell C6100 with Xeon X5650
192 physical cores HT @2666MHz
Interconnection Infiniband QDR 40 Gb/s

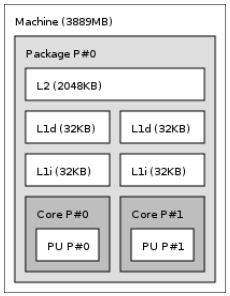
8 nodes HP SL230 with Xeon E5-2667
64 physical cores HT @2666MHz
Interconnection Infiniband FDR 56 Gb/s

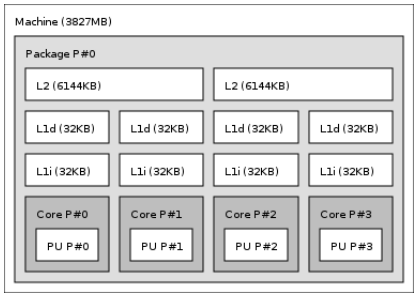
8 nodes Dell R410 with Xeon X5550
64 physical cores HT @2666MHz
Interconnection Infiniband DDR 20 Gb/s

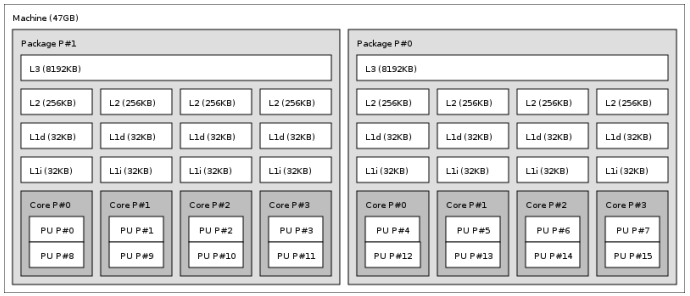
4 nodes Sun X4500 with AMD 285
16 physical cores @2400MHz
Interconnection Infiniband SDR 10 Gb/s

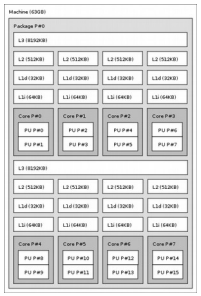


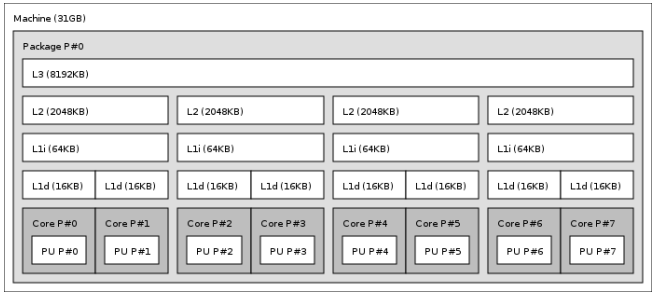
Multicores TechBench
From 2 to 28 cores: examples...

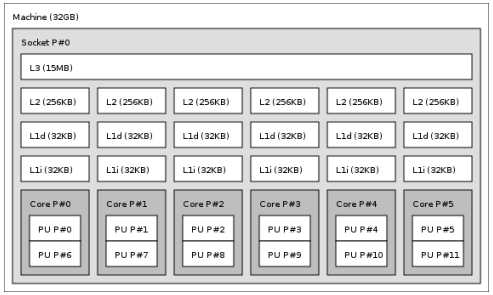


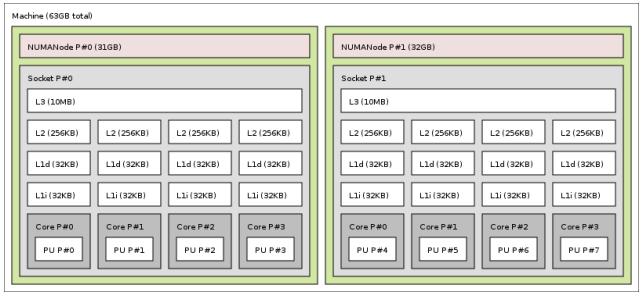


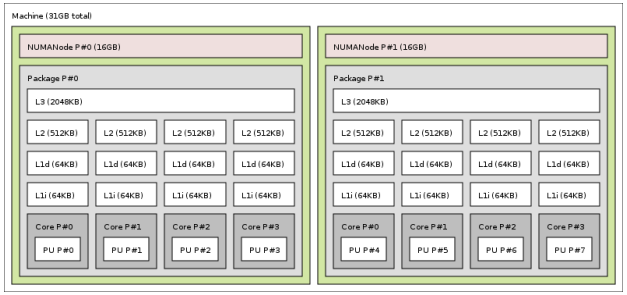


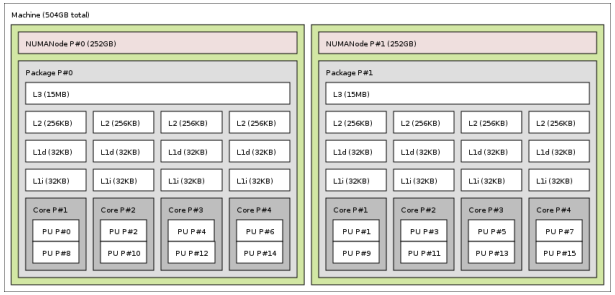












Multishaders TechBench (GP)GPU
84 different models...

GPU Gamer : 26

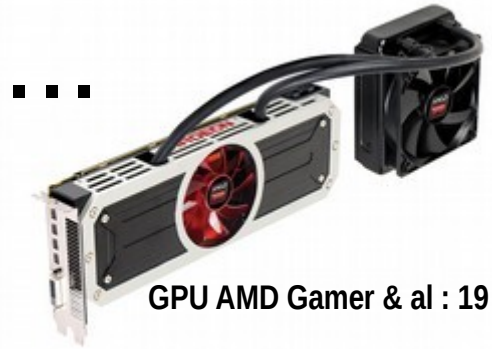
- Nvidia GTX 560 Ti
- Nvidia GTX 680
- Nvidia GTX 690
- Nvidia GTX Titan
- Nvidia GTX 780
- Nvidia GTX 780 Ti
- Nvidia GTX 750
- Nvidia GTX 750 Ti
- Nvidia GTX 960
- Nvidia GTX 970
- Nvidia GTX 980
- Nvidia GTX 980 Ti
- Nvidia GTX 1050 Ti
- Nvidia GTX 1060
- Nvidia GTX 1070
- Nvidia GTX 1080
- Nvidia GTX 1080 Ti
- Nvidia RTX 2070
- Nvidia RTX 2080
- Nvidia RTX 2080 Ti
- ~~Nvidia GTX 1650~~
- Nvidia GTX 1660 Ti
- ~~Nvidia RTX 2060 Super~~
- Nvidia RTX 2070 Super
- Nvidia RTX 2080 Super
- Nvidia RTX Titan

GPGPU : 10

- 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
- Nvidia Volta V100

GPU desktop & pro : 29

- ~~NVS 290~~
- Nvidia FX 4800
- NVS 310
- NVS 315
- Nvidia Quadro 600
- ~~Nvidia Quadro 2000~~
- Nvidia Quadro 4000
- Nvidia Quadro K2000
- Nvidia Quadro K4000
- Nvidia Quadro K420
- Nvidia Quadro P600
- ~~Nvidia 8400 GS~~
- ~~Nvidia 8500 GT~~
- ~~Nvidia 8800 GT~~
- ~~Nvidia 9500 GT~~
- ~~Nvidia GT 220~~
- ~~Nvidia GT 320~~
- Nvidia GT 430
- ~~Nvidia GT 545~~
- Nvidia GT 620
- Nvidia GT 640
- ~~Nvidia GT 710~~
- Nvidia GT 730
- Nvidia GT 1030
- ~~Nvidia Quadro 2000M~~
- ~~Nvidia Quadro K4000M~~
- ~~Nvidia Quadro M1200~~
- ~~Nvidia Quadro M2200~~
- ~~Nvidia MX150~~



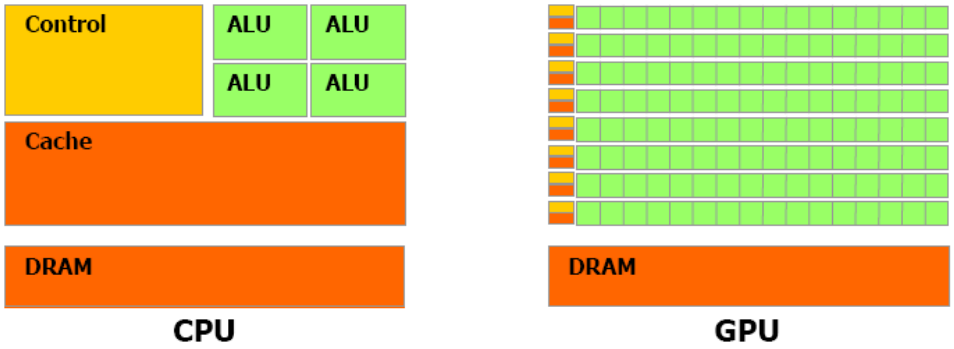
GPU AMD Gamer & al : 19

- ~~HD 4350~~
- ~~HD 4890~~
- ~~HD 5850~~
- ~~HD 5870~~
- ~~HD 6450~~
- ~~HD 6670~~
- ~~Fusion E2 1800 GPU~~
- HD 7970
- FirePro V5900
- ~~FirePro W5000~~
- Kaveri A10-7850K GPU
- ~~R7 240~~
- R9 290
- R9 295X2
- ~~Nano Fury~~
- R9 Fury
- R9 380
- RX Vega64
- ~~Radeon VII~~

Emmanuel Quemener

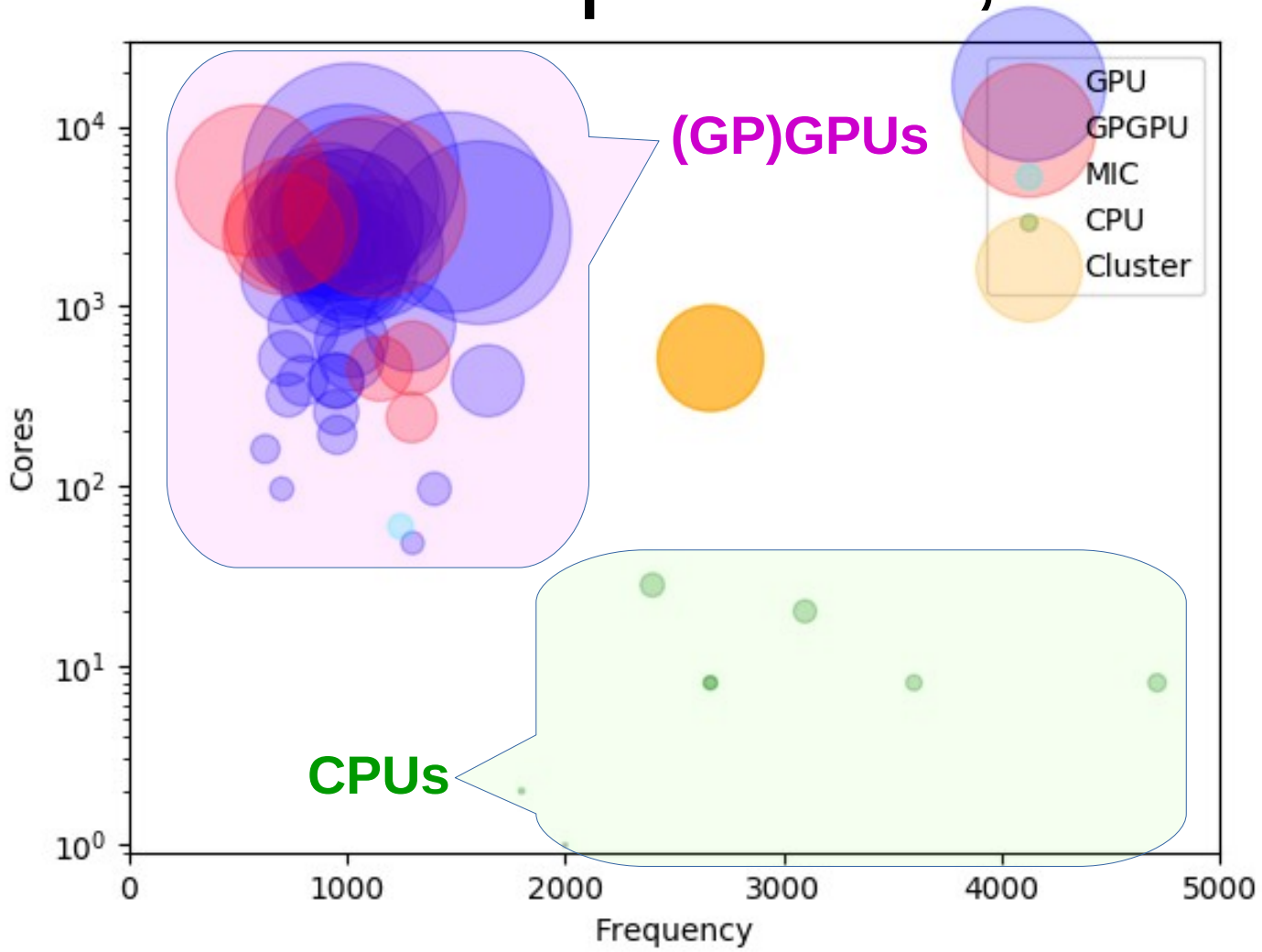
November 15, 2019

From multi-cores to Myri-ALUs ? Differences between GPU & CPU



- Operations
  - Matrix multiplications
  - Vectorization
  - « Pipelining »
  - Shader (multi)processeur
- Programming : 1993
  - OpenGL, Glide, Direct3D, ...
- Genericity : 2002
  - CgToolkit, CUDA, OpenCL

How to represent these testbenchs? Question of frequencies, cores, ...

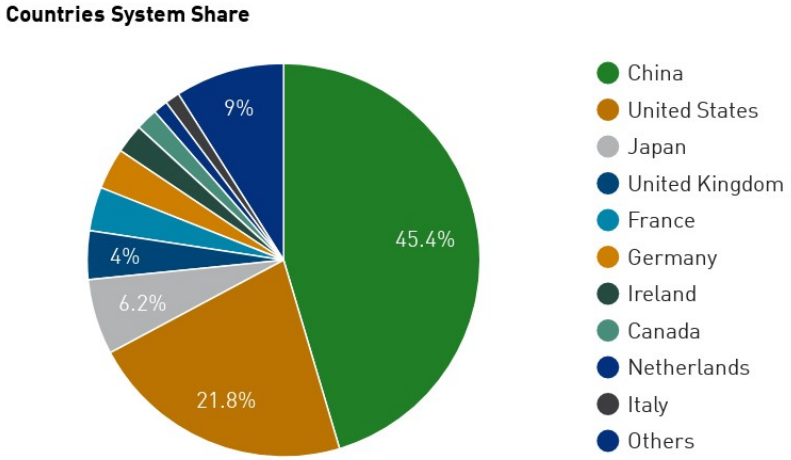


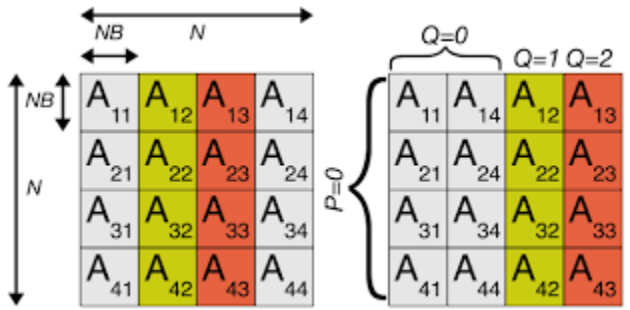
How to compare them ? (Find the best one for my use case)

- Is it possible to find a « *Tolkien code* » ?
  - *One Code to stress them all,*
  - *One Code to evaluate them,*
  - *One Code to confront them all…*
- First approach :
  - Do not reinvent the wheel : use what exists…
- Second approach :
  - Built or migrate one code (at least) with different // paradigms
  - Identify what are the principal properties of the code

Performance of Computers Linpack & the Top 500

- **What** : 500 most powerful super-computer around the world
- **When** : 2 times per year, april & november
- **Where** : around the world
- **Who** (create the code) : ICL from University of Tennessee
- **How** : High Performance LinPack in FP64, LU decomposition for solving Linear Systems
- **How much** : nothing… Open Source code : https://www.netlib.org/benchmark/hpl/
- **Why** : « to flex muscles » between countries...





Why do we NOT use LinPack ? Test by yourself… Scams...

- Too much « free » parameters (never published)
- Too many « scams » from constructors
  - HPL & HPCC : ~ 15 % efficiency
  - Linpack Intel MKL : from 57 % to 92 % efficiency
  - CUDA from Nvidia : ~ 20 % efficiency (and a nightmare to compile)

Robert_Crovella

MODERATOR

Yes, the best HPL performance will come from HPL code specifically provided on a case-by-case basis by NVIDIA. It is not publicly available, and the hpl-2.0_FERMI_v15 will not achieve highest performance on GPUs newer than FERMI.

Posted 02/02/2017 04:18 AM #4

  - How do they reach 75 % on millions cores ?
- Too many bits : only 64 bits, useless for many applications
- So, have a look elsewhere, more **simple**, more **universal**

Build (or migrate) our codes Which properties ? SAROS ones...

- **Simple** : respect the « *Keep It Simple Stupid* » rule
  - The Core code is focused on the main involving processes
  - The « grain » and « bound » problems are well identified
- **Agnostic** : can be launched on all devices, all OS
- **Reproducible** : in space & time
- **Open Source** : shared for every one
- **Small** : few dependencies to software & small space print
- Can retreive mine for *PU via subversion :
  - svn checkout https://forge.cbp.ens-lyon.fr/svn/bench4gpu/

Code : develop, integrate, migrate ?

Developping (from scratch)

A huge mass (project)...
Successive fining... (code)...
To a real product... (application).

Integrate

Some components (*framework*)...
... by assembling...
... To a real product (application).

How to program in // ?
The different librairies

	Cluster	Node CPU	Node GPU	Node Nvidia	Accélerateur
MPI	Yes	Yes	No	No	Yes*
PVM	Yes	Yes	No	No	Yes*
OpenMP	No	Yes	No	No	Yes*
Pthreads	No	Yes	No	No	Yes*
OpenCL	No	Yes	**Yes**	**Yes**	Yes
CUDA	No	No	No	**Yes**	No
TBB	No	Yes	No	No	Yes*
OpenACC	No	Yes	No	Yes	Yes
Kokkos	No	Yes	No	Yes	Yes

Parallel programming integration libraries...

	Cluster	Nœud CPU	Nœud GPU	Nœud Nvidia	Accélerateur
BLAS	BLACS MKL	OpenBLAS MKL	clBLAS	CuBLAS	OpenBLAS MKL
LAPACK	Scalapack MKL	Atlas MKL	clMAGMA	MAGMA	MagmaMIC
FFT	FFTw3	FFTw3	clFFT	CuFFT	FFTw3

And my « pure » code ?

- OpenACC (inside PGI) :
  - A « pragma »tic approach looking like OpenMP
  - Fully functional which need « only » some modifications in codes
- Kokkos :
  - Another « pragma »tic approach looking like OpenMP
  - More restrictive to code (relative to OpenACC)
- Par4All (orphaned but promising) :
  - A preprocessor analyses the code and translate it
  - 3 implementations : OpenMP, Cuda and OpenCL
  - Educational project but impossible to use now (compiler version)

Matrix Matrix Multiplication : Why a GPU so « efficient » ?

- Consider 2 matrices A & B of dimensions NxP and PxM
- The matrix product of A by B gives C
- Each element of C, C_{ij} for i from 1 to N & j from 1 to M :

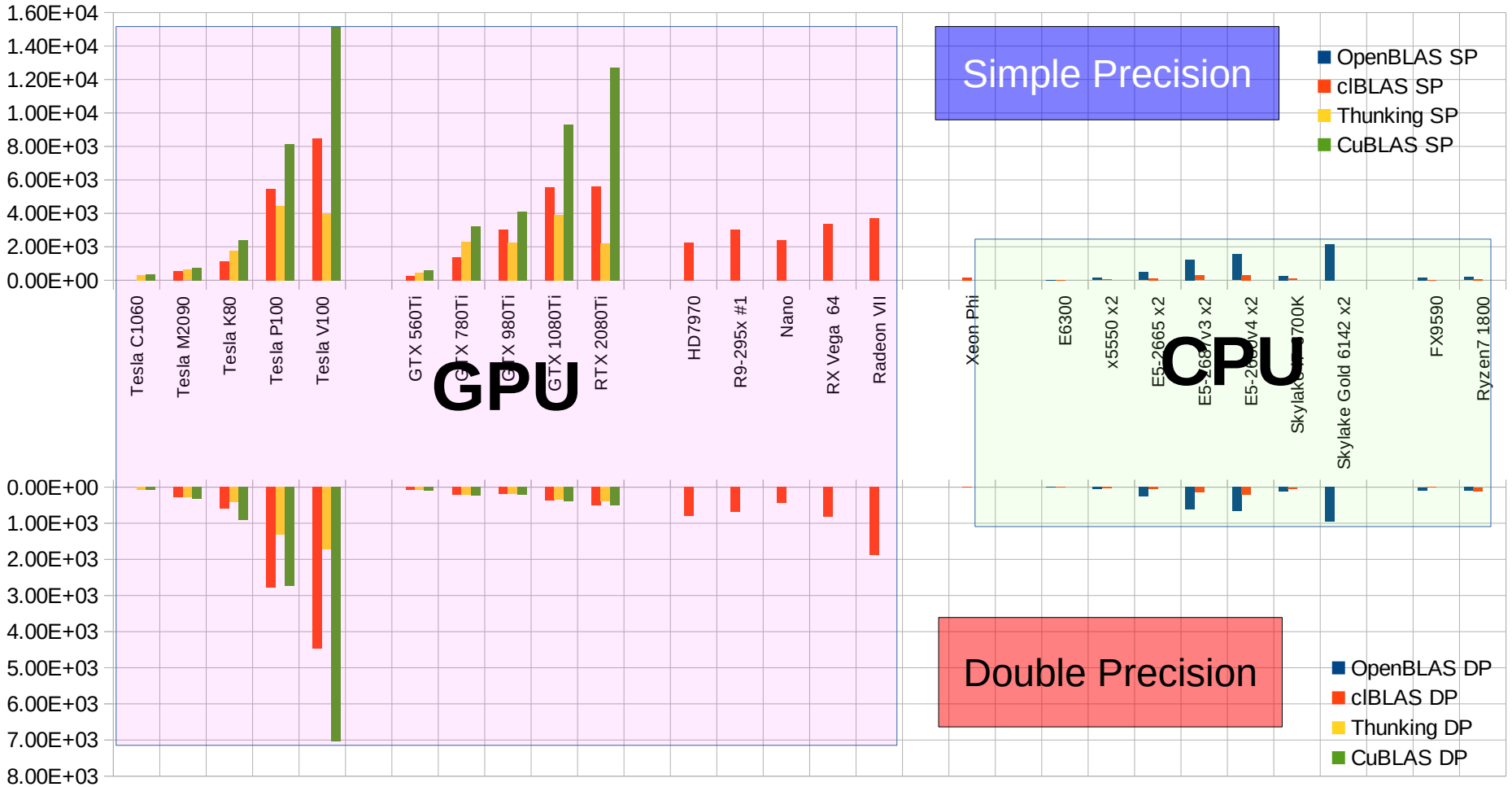
$$C_{ij} = \sum_{k=1}^{k=P} A_{ik} B_{kj}$$

- The C_{ij} are independant : « coarse grain » parallelism
- The $A_{ik}B_{kj}$ groupable by blocks : **vectors & FMA units**

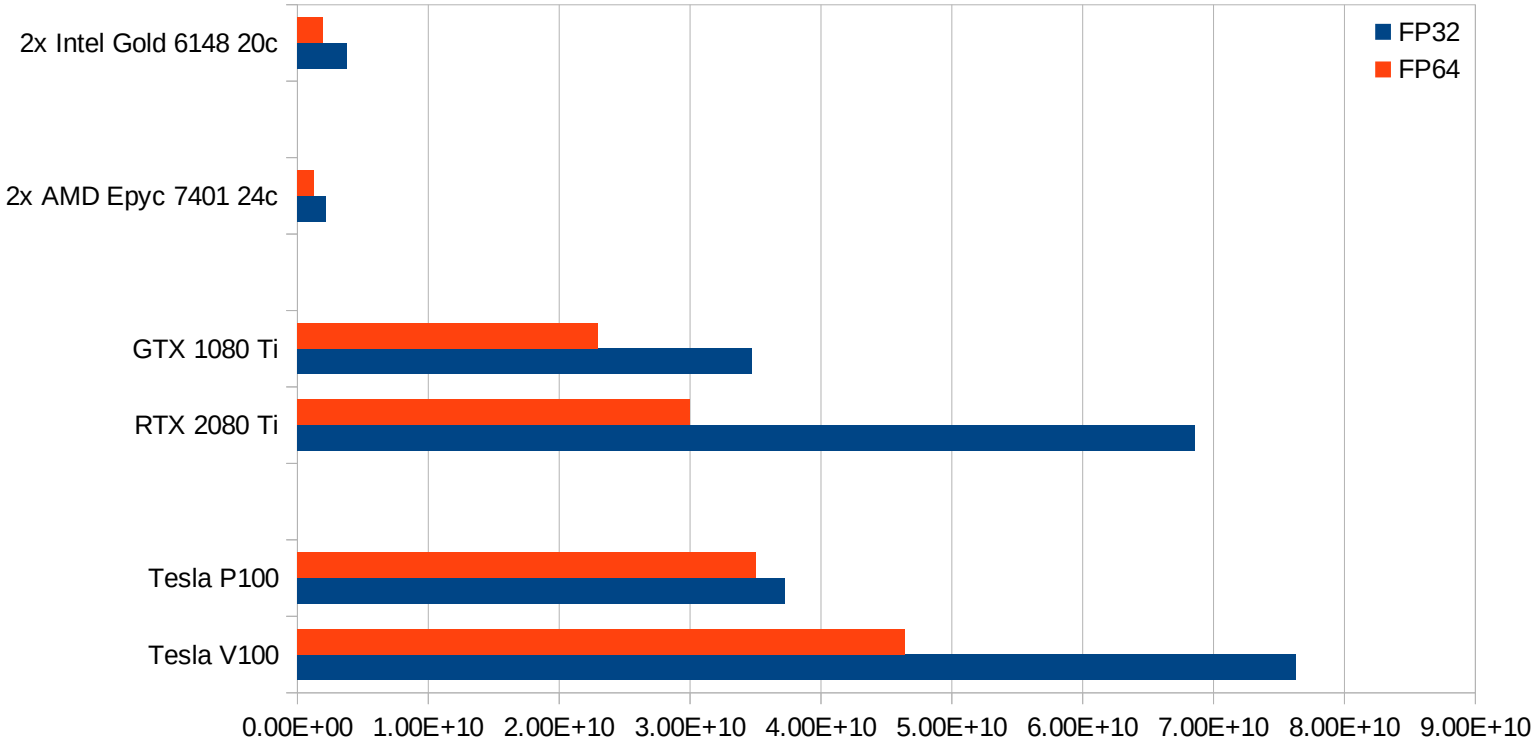
First low level « integration » Let's discover BLAS

- Fonctions BLAS
  - 3 levels of fonctions :
    - Level 1 : rotations, normes, swaps, copies, scalar product, ...
      - Exemple : xSWAP t exchange 2 vectors
    - Level 2 : matrix-vector product, resolution of triangular systems,
      - Exemple : xTSRV to solve the triangular system
    - Level 3 : simple operations of matrices
      - Exemple : xGEMM to multiply matrices
- The functions of all our attentions :
  - sGEMM & dGEMM for the product simple & double precision

Matrix Matrix Product Really So powerful ? **x7**



With a extended BLAS process… On 6 CPU, GPU, GPGPU benchs.



- 5 BLAS functions used : TRSV, GEMV, AXPY, NRM2, SWAP
- Sizes from 1024 to 46080
- Performance criteria : (size*size)/ElapsedTime
- **Speedup around 20 and 35**

Going down a new step « Developer » approach

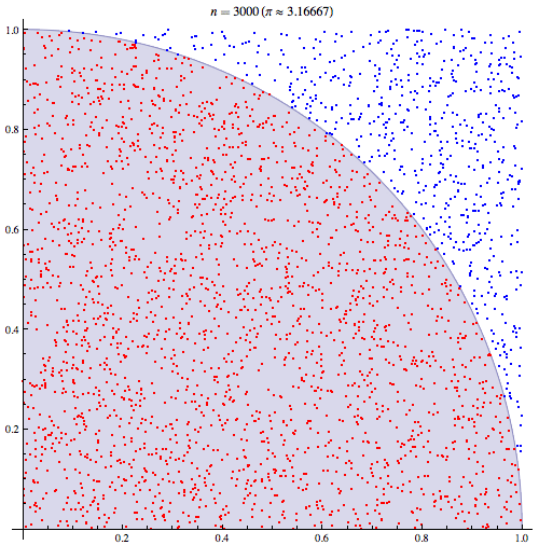
- On my experiences :
  - « Business » code : speed'up from 2 to 5
  - « integrator » approach : with optimized librairies, factors 7x to 35x
- Now, which codes to « touch the beast » :
  - A « coarse grain » code, code « ALU » : **Pi Dart Dash**
  - A « fine grain », code « memory » : **Nbody**
  - A code « memory grain » : **Splutter**
- 2 goals : to compare GPU vs CPU & GPU vs GPU

OpenCL environment : C but 13 implementations on x86

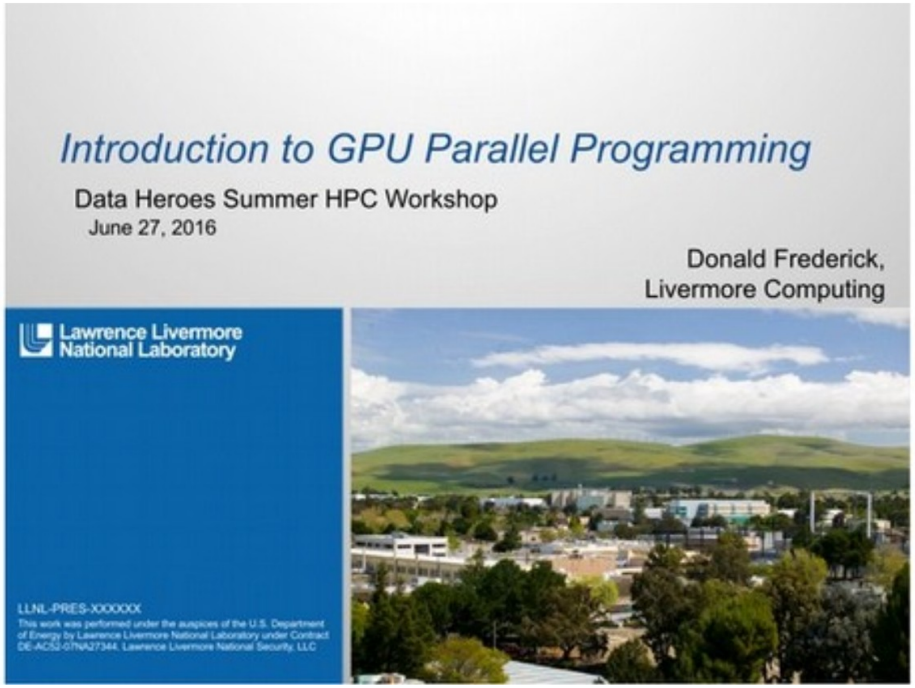
- 3 implementations for CPU:
  - AMD one : the historical one, works in all conditions, like a good OpenMP use
  - Intel one : very efficient (but not always, for recent processors and AMD one)
  - OpenSource one : PortableCL (POCL)
- 6 implementations for GPU:
  - AMDgpu-pro : for their recent GPU, very tricky to use...
  - Nvidia: only for the eponymous GPU
  - Beignet : Open Source version for the Intel GPU (integrated besides CPU on socket)
  - Intel : new version for IRIS Intel GPU
  - Mesa : Open Source version, essentially for AMD, for recent GPU but not the latest
  - ROCm : Open Source version, very promising, but for very recent systems
- 1 implementation for Accelerator:
  - Intel's: for the MIC Xeon Phi Knight Corner
- 2 implementations for FPGA :
  - Altera, now provided by Intel
  - Xilinx
- 1 implementation for DSP :
  - Texas Instruments

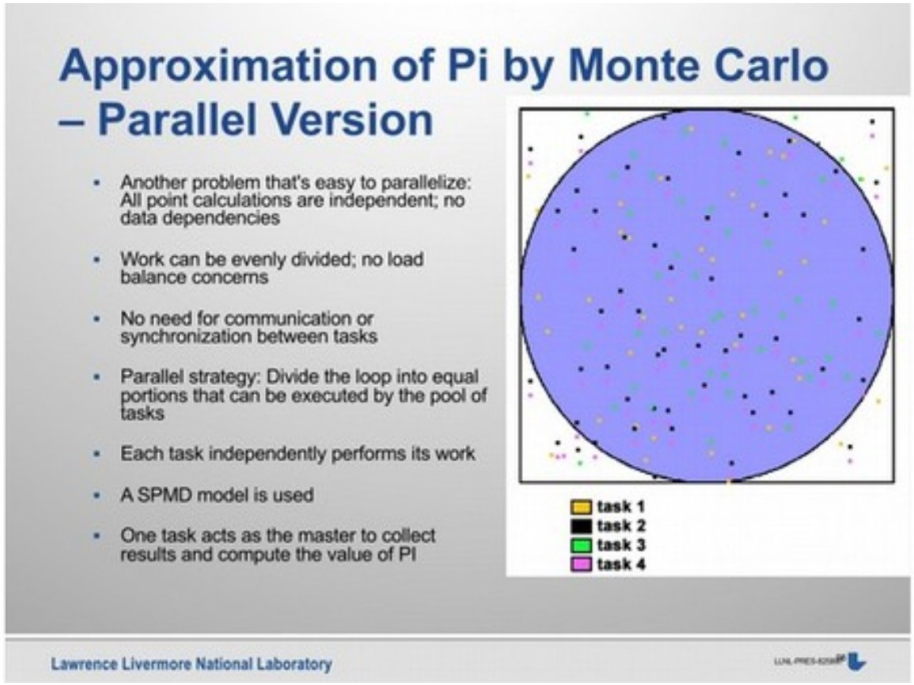
And for a simple implementation ?
PiMC : Pi by Dart Board Method

- Classical illustration of Monte Carlo method
- Parallel implementation : distribution of iterations
  - From 2 to 4 parameters
    - Total number of iterations
    - Parallel Rate (PR)
    - (Type of variable : INT32, INT64, FP32, FP64)
    - (RNG : MWC, CONG, SHR3, KISS)
  - 2 simples observables :
    - Pi estimation (just indicative, Pi is not rationnel :-))
    - Elasped time



Not very relevant as code
Just have a look to a LLNL tutorial...

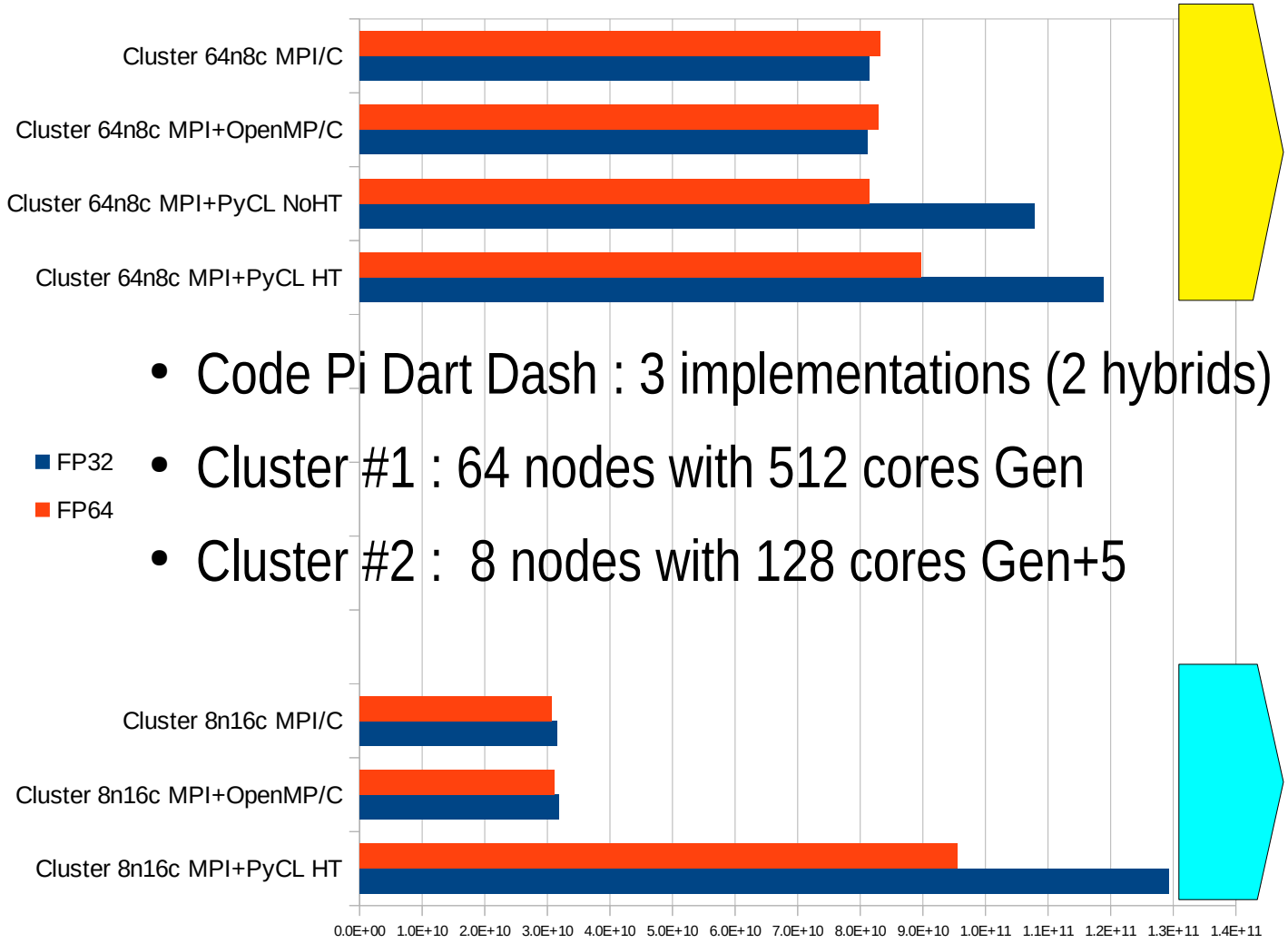




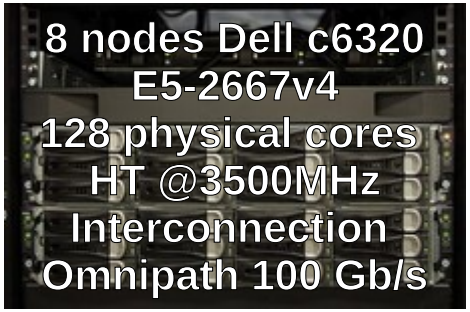
A very monolithic version… Variable & RNG fixed…

```
unsigned int jcong=seed_z+i;
#define CONG  (jcong=69069*jcong+1234567)
#define CONGfp CONG * 2.32830643545494e-10f
LENGTH total=0;
for (LENGTH i=0;i<iterations;i++) {
        float x=CONGfp ;
        float y=CONGfp ;
        unsigned long inside=((x*x+y*y) < 1.0f) ? 1:0;
        total+=inside;
}
Inside(i)=total;
```

On clusters ? Python efficient ? A small comparison with GNU

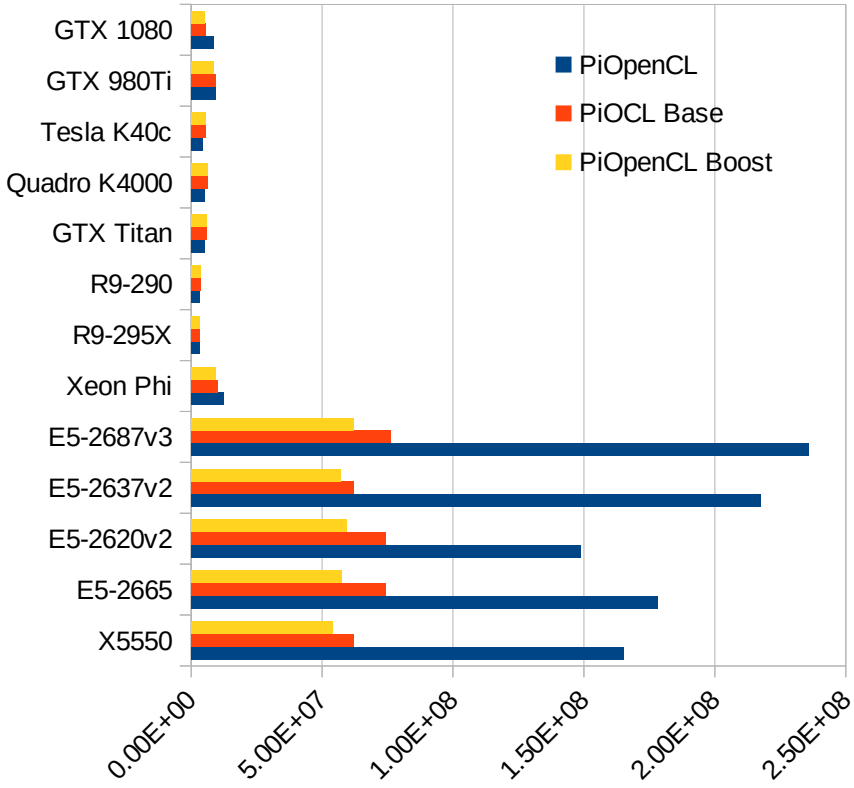


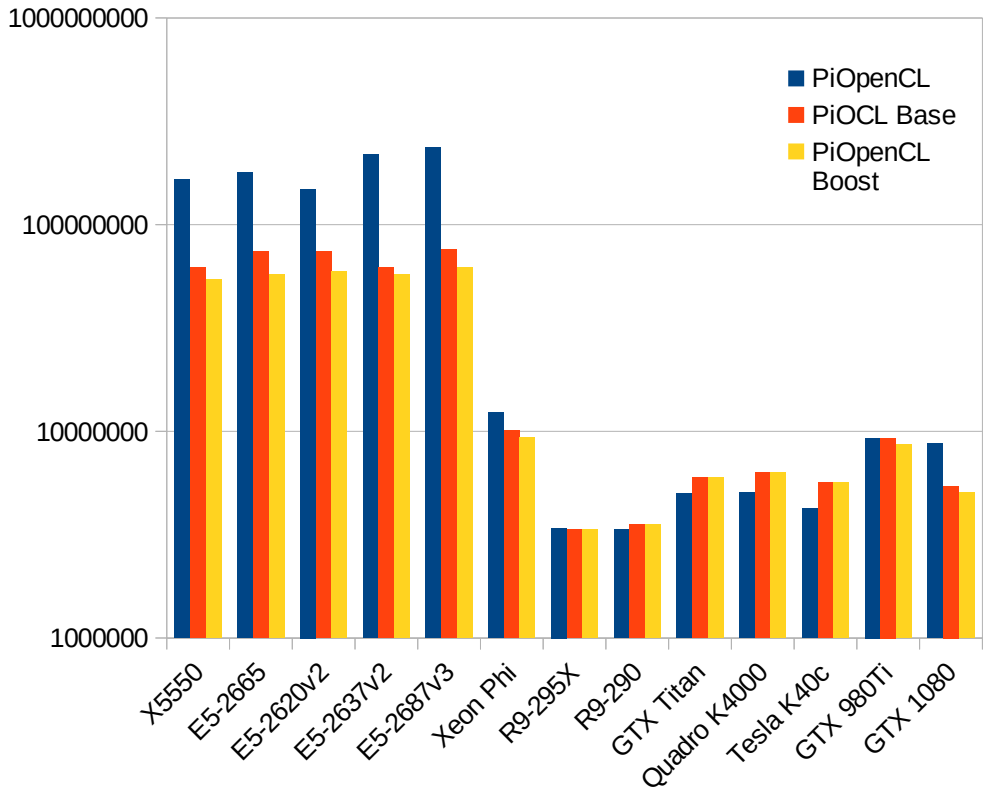
- Code Pi Dart Dash : 3 implementations (2 hybrids)
- Cluster #1 : 64 nodes with 512 cores Gen
- Cluster #2 : 8 nodes with 128 cores Gen+5



For PR of 1, low rate, Pi
The CPU explodes the GPU!

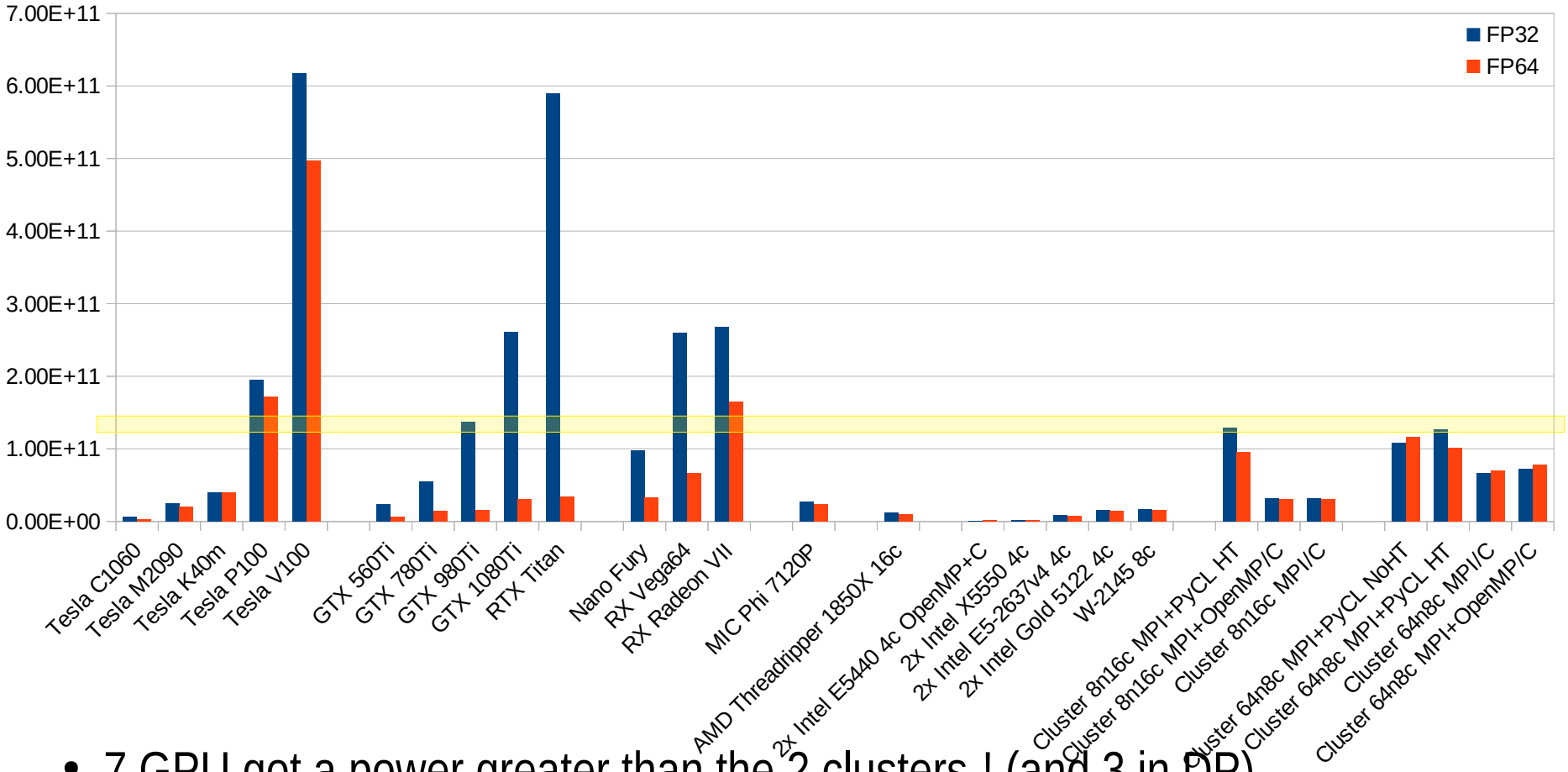
From 20 to 50x slower !





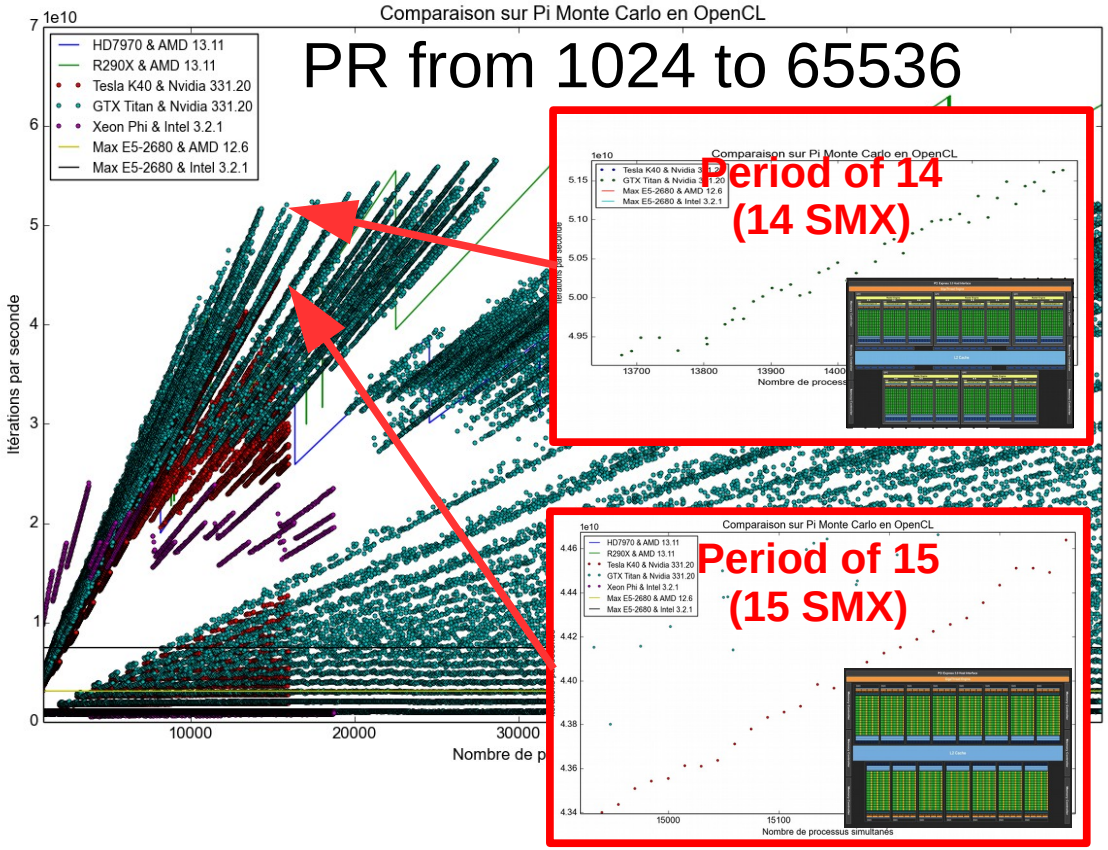
1.5 order of magnitude

Pi Dart Dash, clusters added… Very impressive !

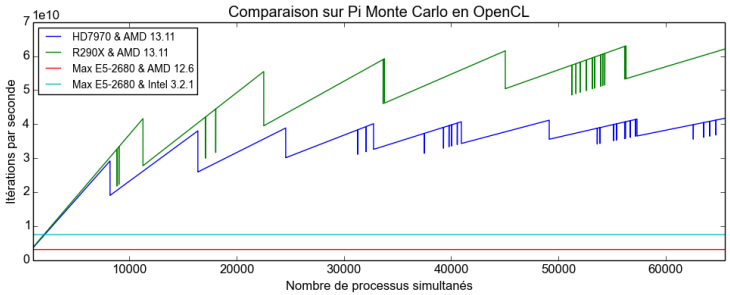


- 7 GPU got a power greater than the 2 clusters ! (and 3 in DP)
- Speed up with best CPU : **around 35x**

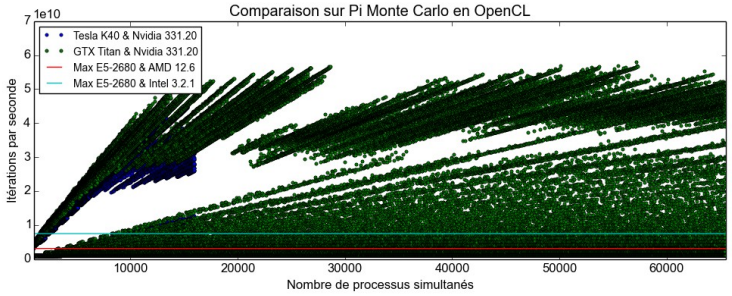
Investigate the internal structure By the execution Pi Dart Dash



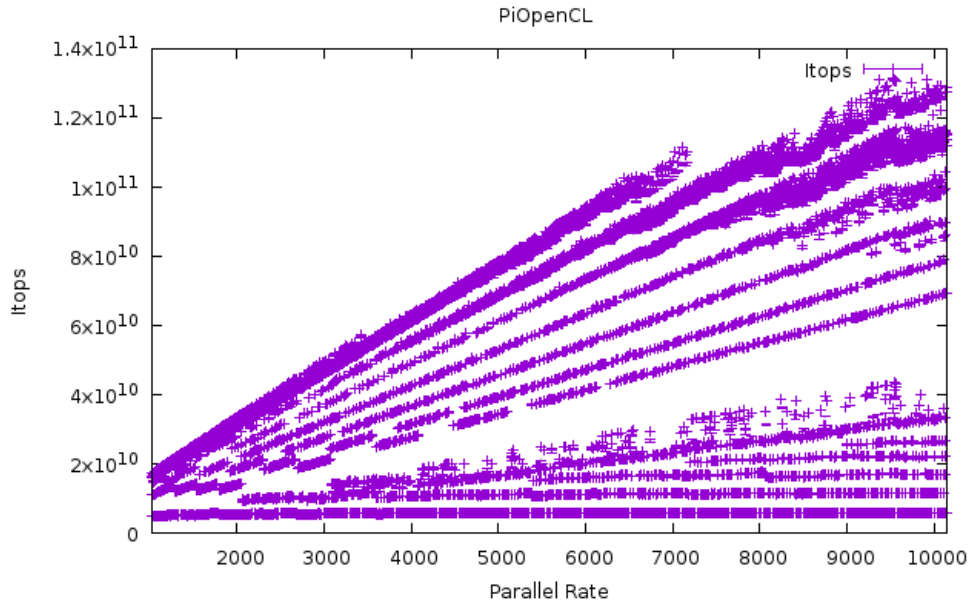
AMD HD7970 & R9-290
Long period : 4x number of ALU

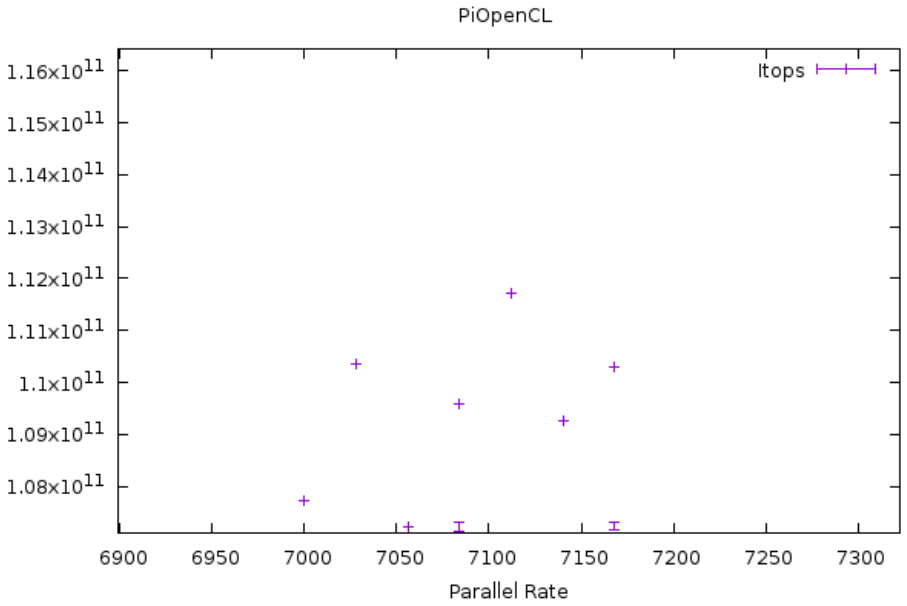


Nvidia GTX Titan & Tesla K40
Small Period : number of SMX unit



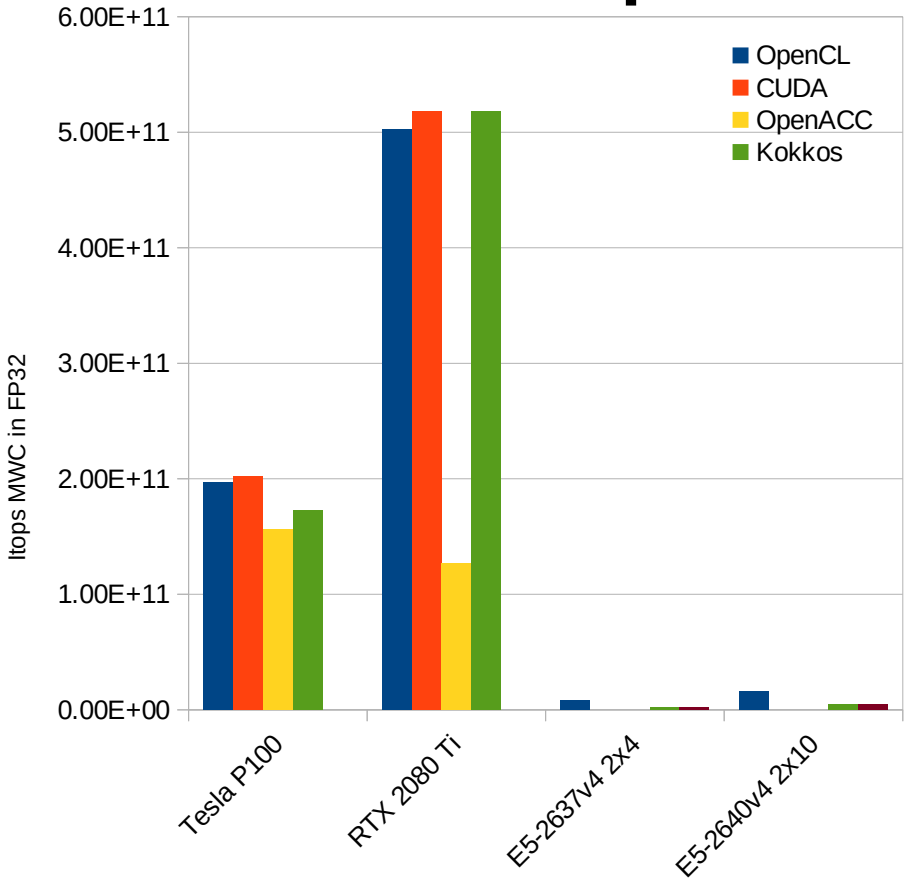
And if we use a OpenCL code written in C only ?

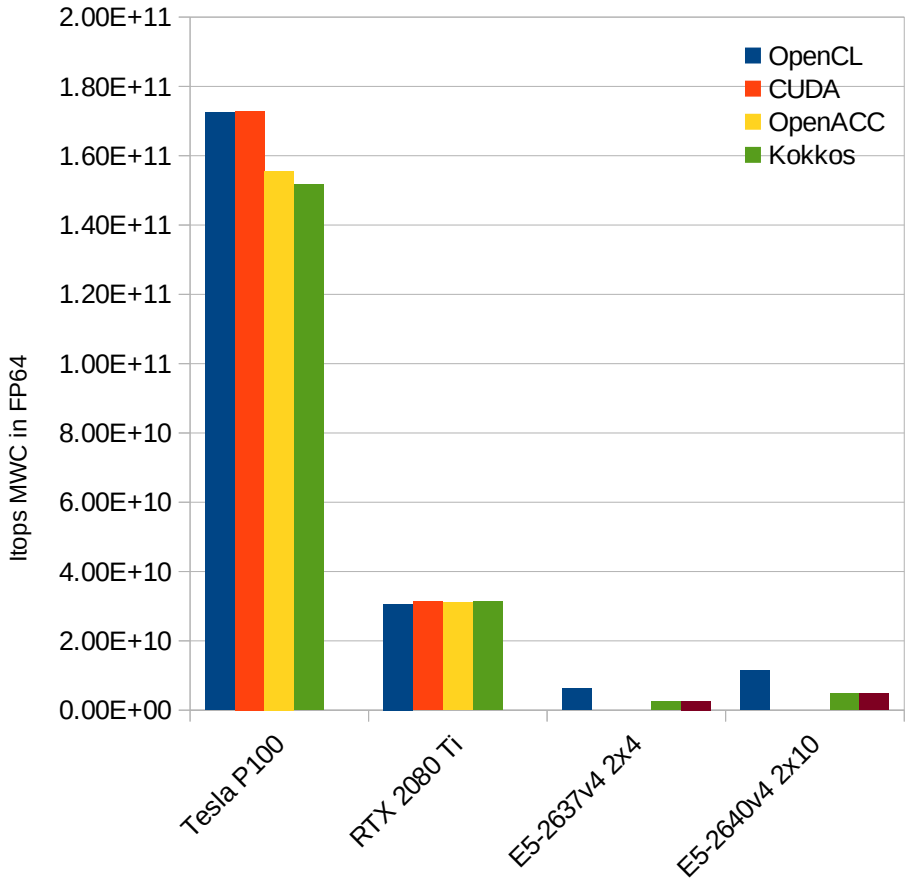




- Same detection of prime numbers
- Same local maximum (xN the number of cudacores)

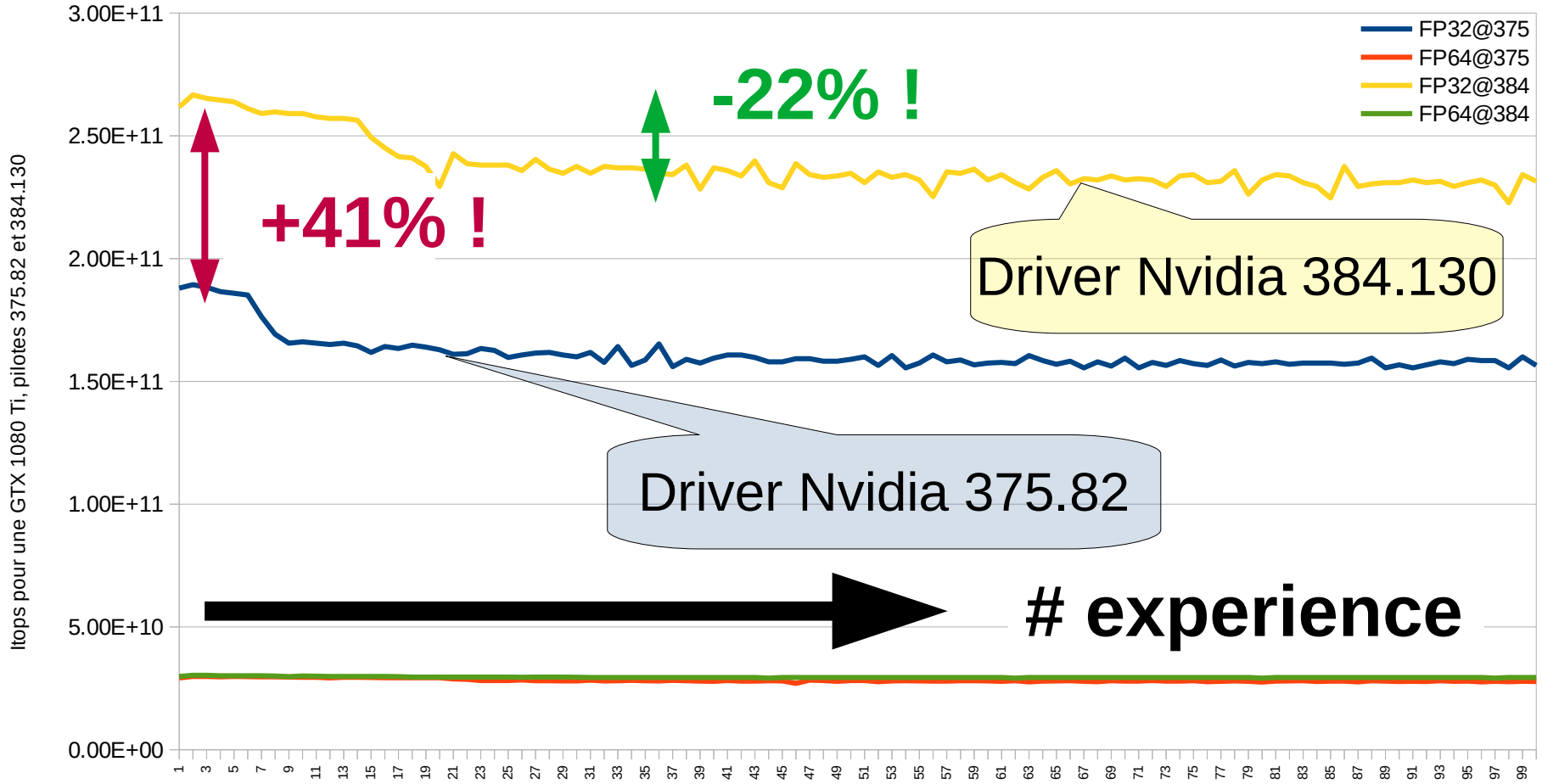
And for the other ways for GPU CUDA, OpenACC & Kokkos ?





Finally, comparable… What to evaluate : entry cost..

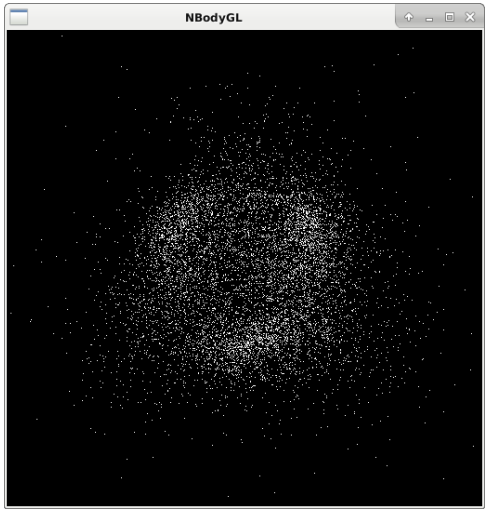
Versions of drivers & experiences Variability factors...

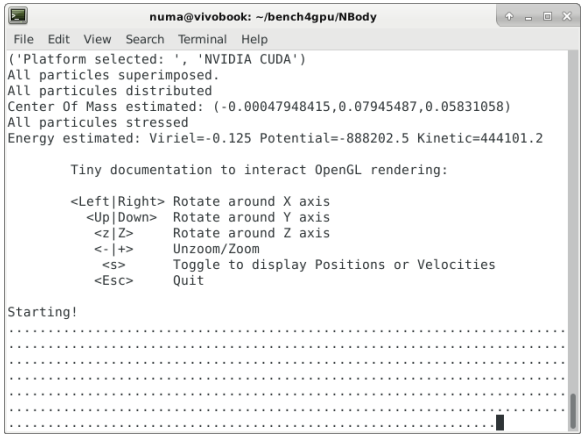


So retreive as many informations as possible...

Back to physics
A newtonian N-Body code...

- Let's use « fine grain » code
- Code N-body very simply implemented
  - Second Newton's law, autogravitating system
- Differential integration methods :
  - Implicit Euler, Explicit Euler, Heun, Runge Kutta
- Input parameters :
  - Physical ones : number of particules
  - Numerical ones : integration step, number of step
  - Architecture : computing in FP32 or FP64





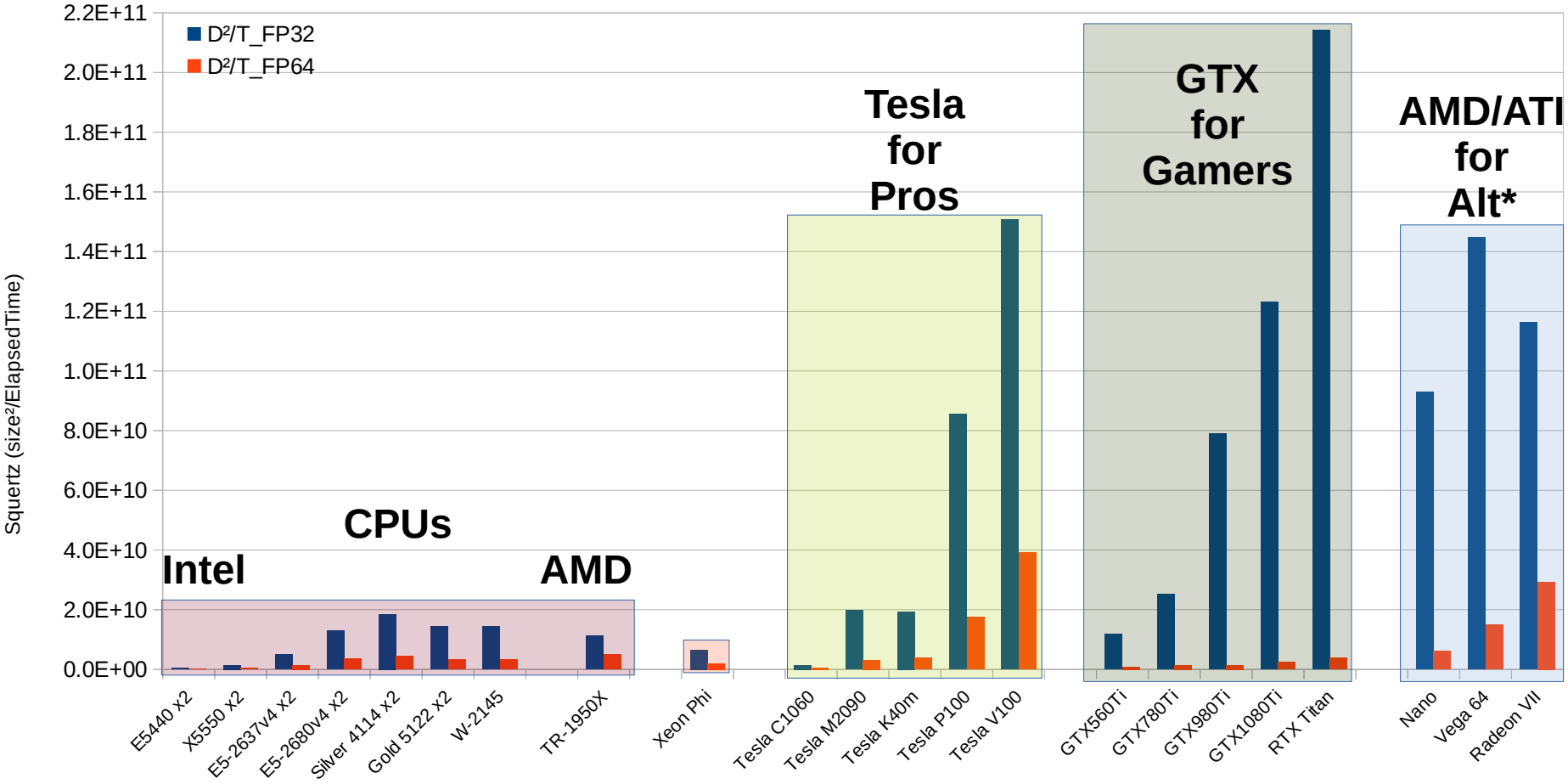
Nbody : as simple as I can !

- Languages : Python, OpenCL & OpenGL
  - Python as glue, OpenCL for *PU programming, OpenGL for rendering
- Newtonian approach : second law used
- Language : Python & OpenCL
- Positions & velocities : float4 or double4 used, from OpenGL
  - (x,y,z,0) as and (v_x,v_y,v_z,0)
- Simple definition of force or potential
- Resolution via :
  - Implicit Euler, Explicit Euler, Heun, Runge Kutta

Little demonstration for NBody ?

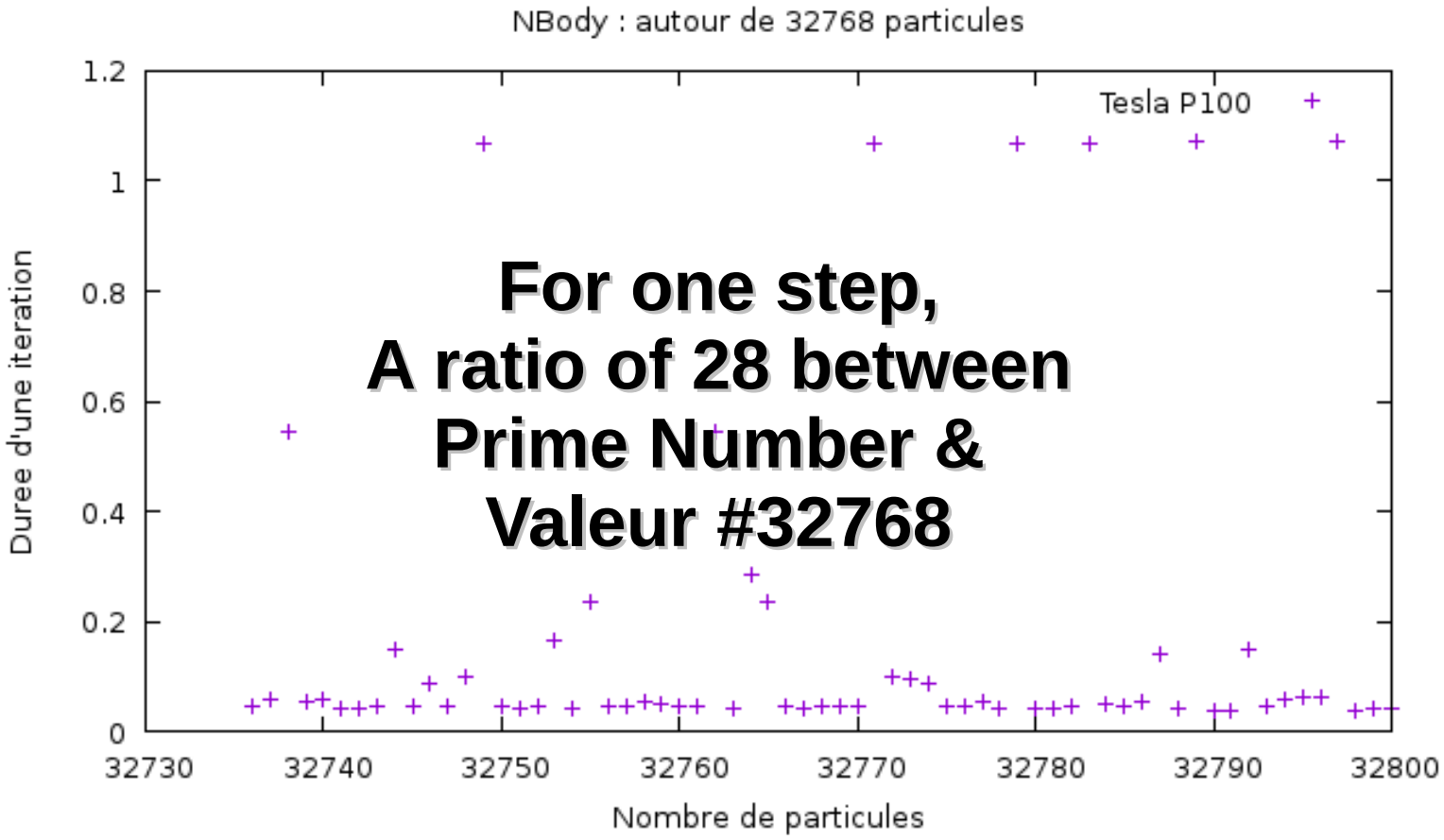
- Several use cases :
  - 8192 particules in FP32
  - 8291 particules in FP32 (8191 is prime!)
  - 8193 particules in FP32
  - 8192 particules in FP64
- Inside GPU code
  - The Python call for kernels
  - The managment of coordonates
  - The use of numpy for OpenGL rendering

A « naive » N-Body code, 32768p Between the bests…



Speedup from **8 to 12** with best CPU system...

And the « prime numbers » effect for Nvidia GPU ?

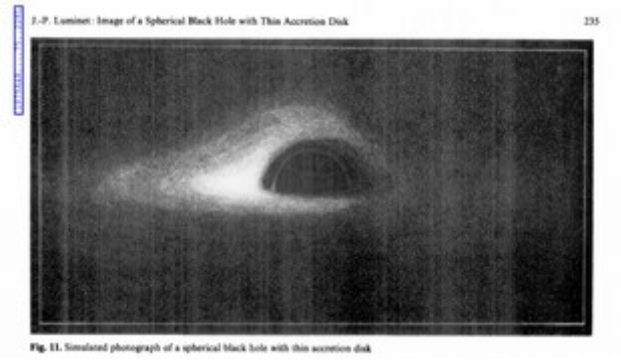


SAROS codes on *PU
Little summary of best speedups

- Integration :
  - BLAS xGEMM : **7x** in FP32 and in FP64
  - BLAS xTSRV : **20x** in FP32 and in FP64
- Development :
  - Pi : coarse grain code, around **36x** in FP32, around **32x** in FP64
  - Nbody : fine grain code, around **11x** in FP32, around **9x** in FP64
- Speed up :
  - Great difference between FP32 and FP64 on GPU a GPGPU systems
  - AMD GPU : very good ratio performances/price, but hard to use !

40y between simulation & measure
From 2019 to 1979...

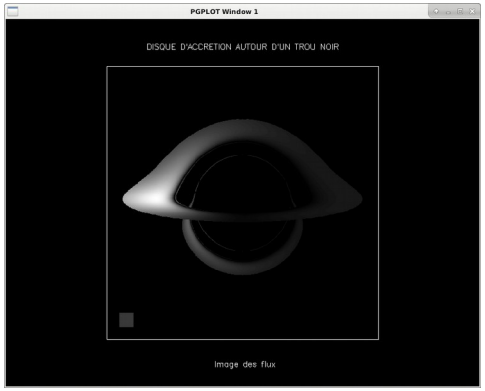
1979 : JP Luminet A&A



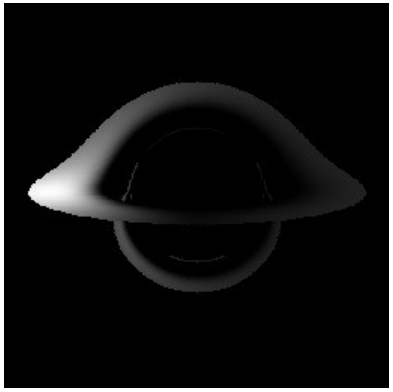
2014 : Film Interstellar

2019 : EHT, Messier 87

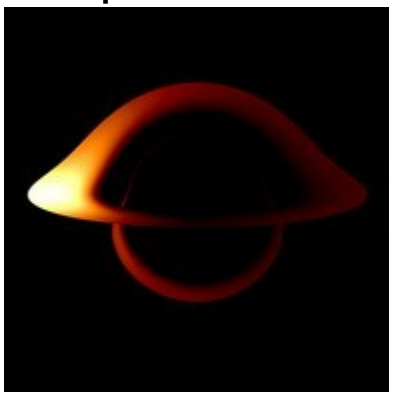
1994 : Fortran Code



1997 : C Code



2019 : OpenCL/CUDA



Back to basics
Everything inside Luminet article !

- General relativity of Einstein
- Schwarzschild tmetric
- Polar reduction equation
- Differenciation of polar equation
- Second order system
- Emission model of disc
  - Monochromatic frequency : school case
  - Black Body : more realistic model

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 secundary 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:

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

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

From article to report

- Schwarzschild Metric :

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

- Polar équation :

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

- Coordonnates change : *u=1/r*

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

- Derivation of polar equations :

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

- Equation system to solve : $v = \frac{du}{d\phi}$ and $\frac{dv}{d\phi} = 3\frac{m}{b}u^2 - u$

DISQUE D'ACCRETION
AUTOUR D'UN TROU NOIR

IMAGE & SPECTRE

Hervé Aussel
Emmanuel Quémener

Travail réalisé dans le cadre des travaux pratiques de
"Modélisation numérique"

Rapport du 2 mars 1994

Plasma scarf around Black Hole
Pas « sans » mais « avec » dessus-dessous...

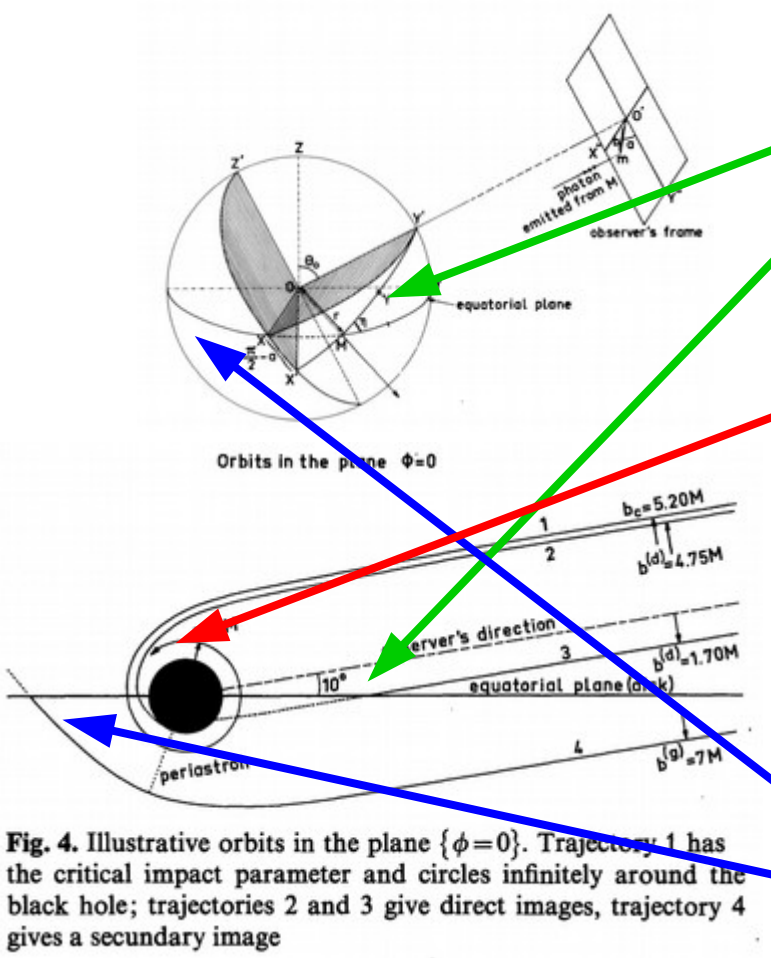
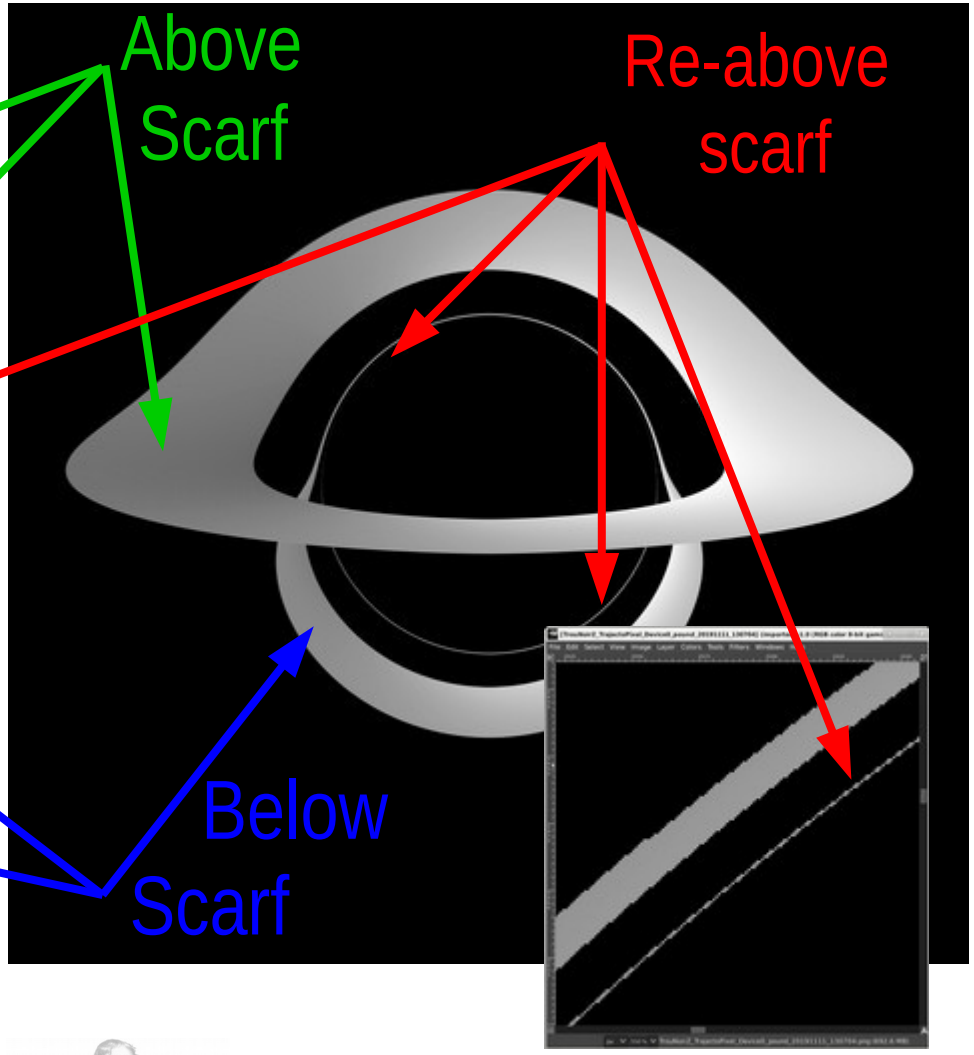


Fig. 4. Illustrative orbits in the plane $\{\phi=0\}$. Trajectory 1 has the critical impact parameter and circles infinitely around the black hole; trajectories 2 and 3 give direct images, trajectory 4 gives a secundary image



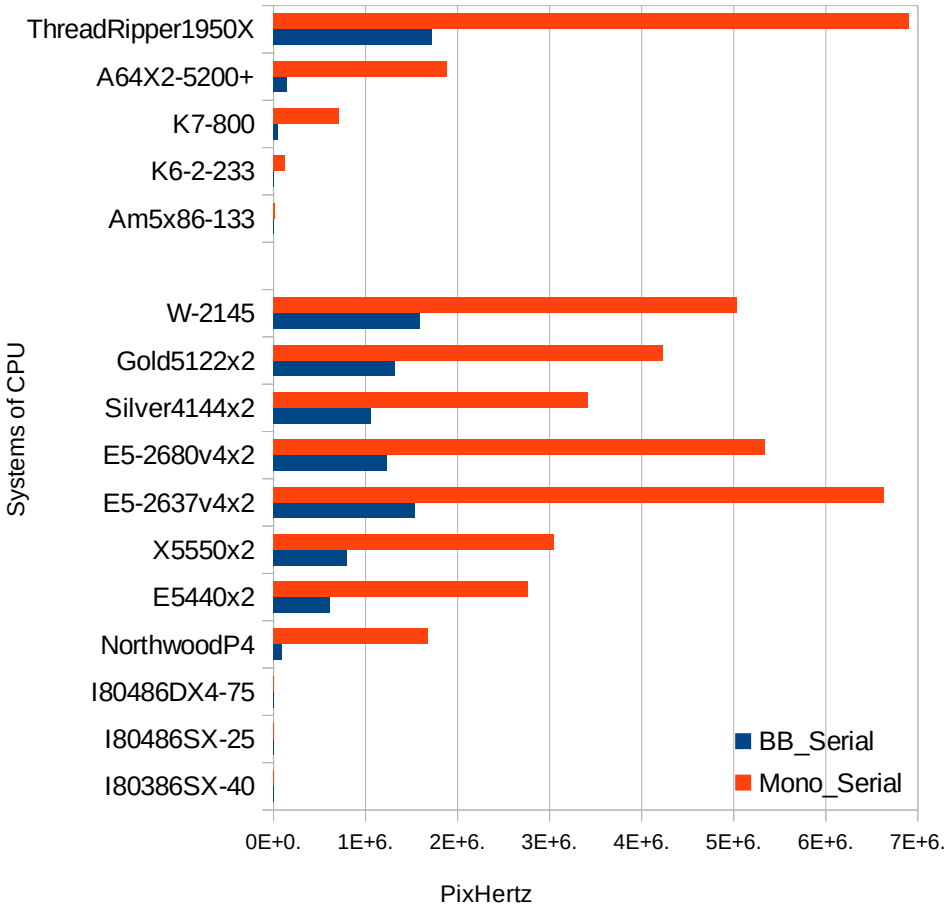
« économique » : use the cylindric symmetry

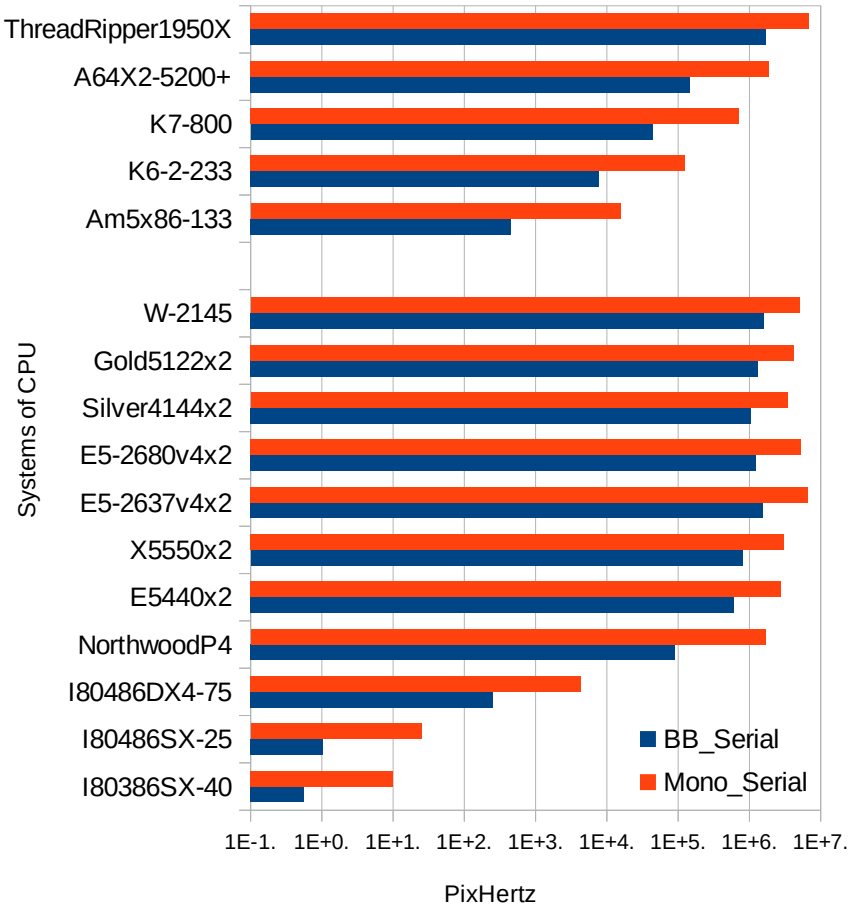
- For each « impact parameter » (distance to center)
  - Compute the trajectory of photon for observer plane
  - For each pixel with this impact parameter :
    - Estimate the interception index corresponding to disc angle
    - Test if the distance to centre this index is between internal and external radius
      - Estimation of Doppler & Einstein effects
      - Estimation of flow by two methods :
        - Monochromatic emission : simple but instructive
        - Black Body emssion : more realistic with Planck Spectrum
- Much more efficient & Computing ~ #pixels
  - Exploitation of PixHertz (number of pixels over Elapsed time)...

Test bench for 16 CPUs

- The processors and their 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 from 64x64 to 16384x16384 pixels : 2**6 à 2**14
  - Except for the very very old CPU : limitation to 256x256
- Methods for impact : 2 to explore with different computing « loads » différentes
  - Low computing load : « Monochromatic » (ak Mono)
  - High computing load : « Black Body » (aka BB)
- Statistics : 10 successive launchs
  - Use of the median for the « Elapsed Time »

Execution of sequential code We win in 30 years

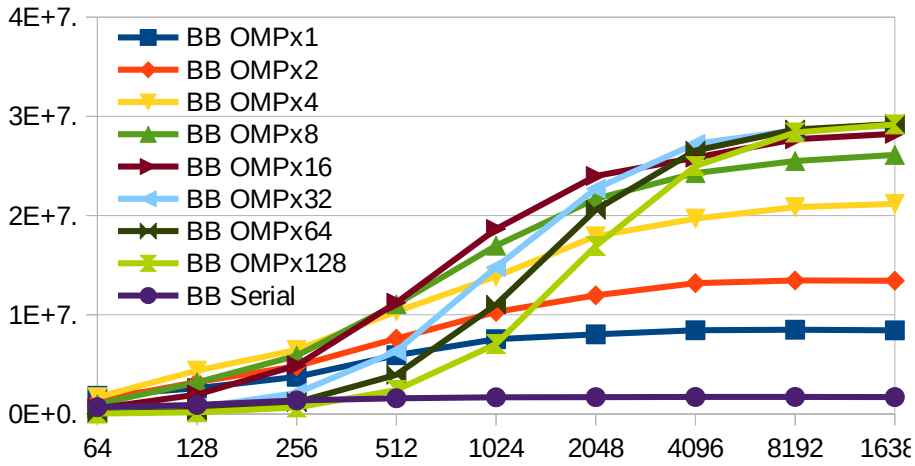


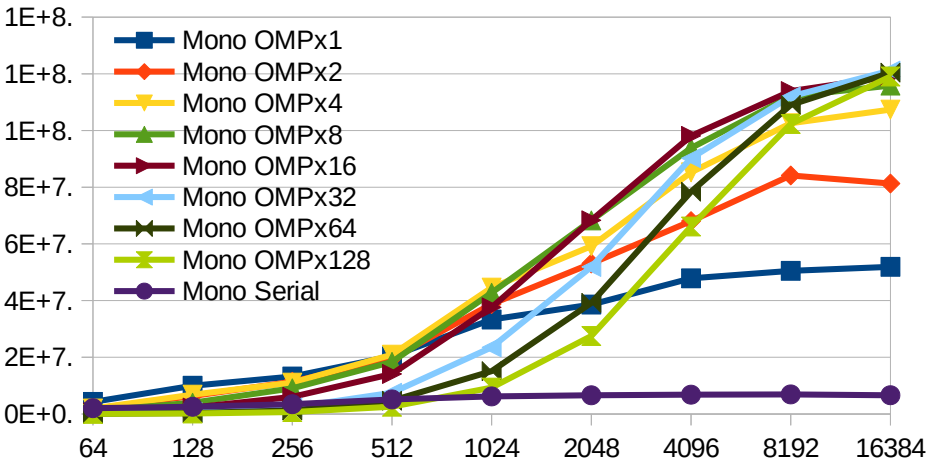


Gain Best/Worse : 3 millions in BB & 700000 in Mono...

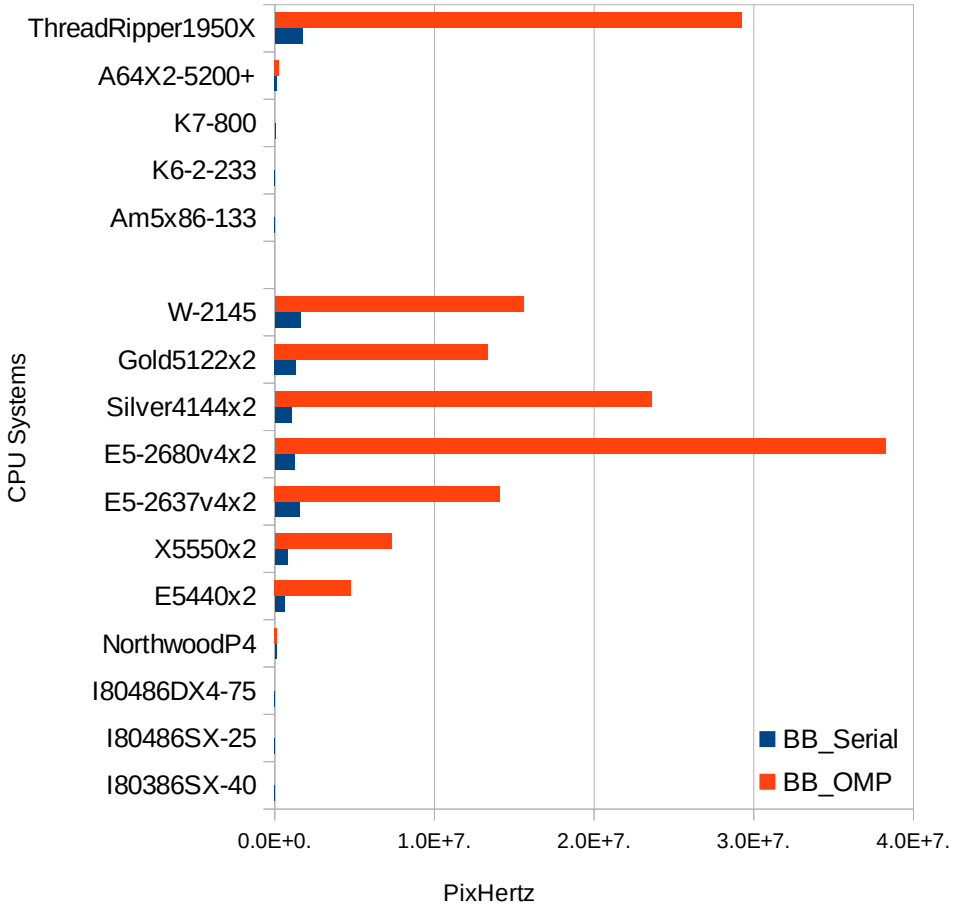
Parallelization of code Migration in OpenMP & strangies

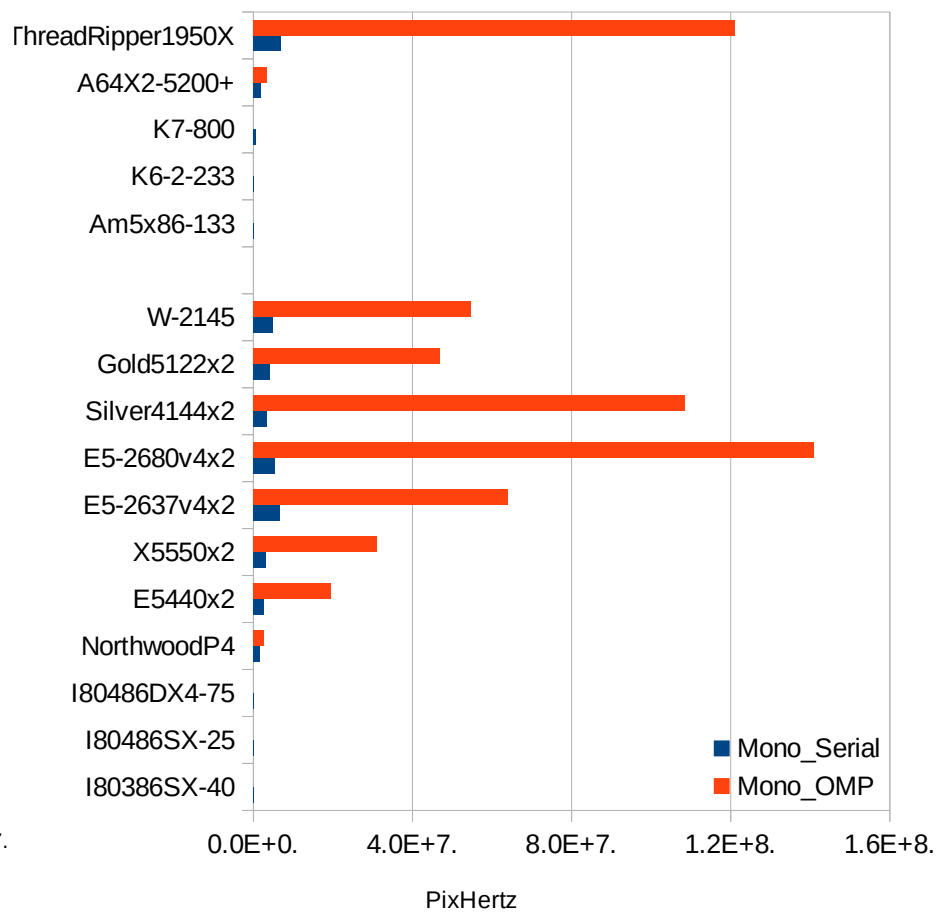
- « Natural » parallelization : impact parameter
  - Minor modification of code (move declaration of variables)
  - Fear: computing load not equal between different threads
  - Exploration of different « OMP_NUM_THREADS » : from 1x to 128x
- For the best : ThreadRipper 1950x, a **x17-x18**





Parallelization in OpenMP
We win more than expected in BB !



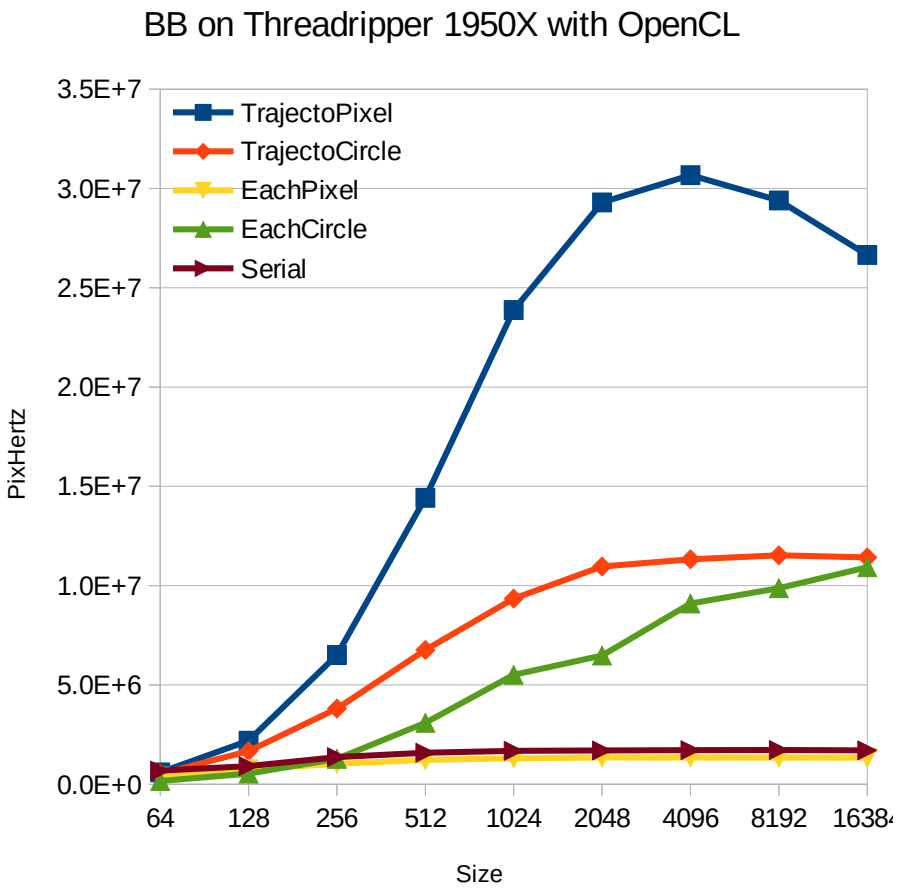


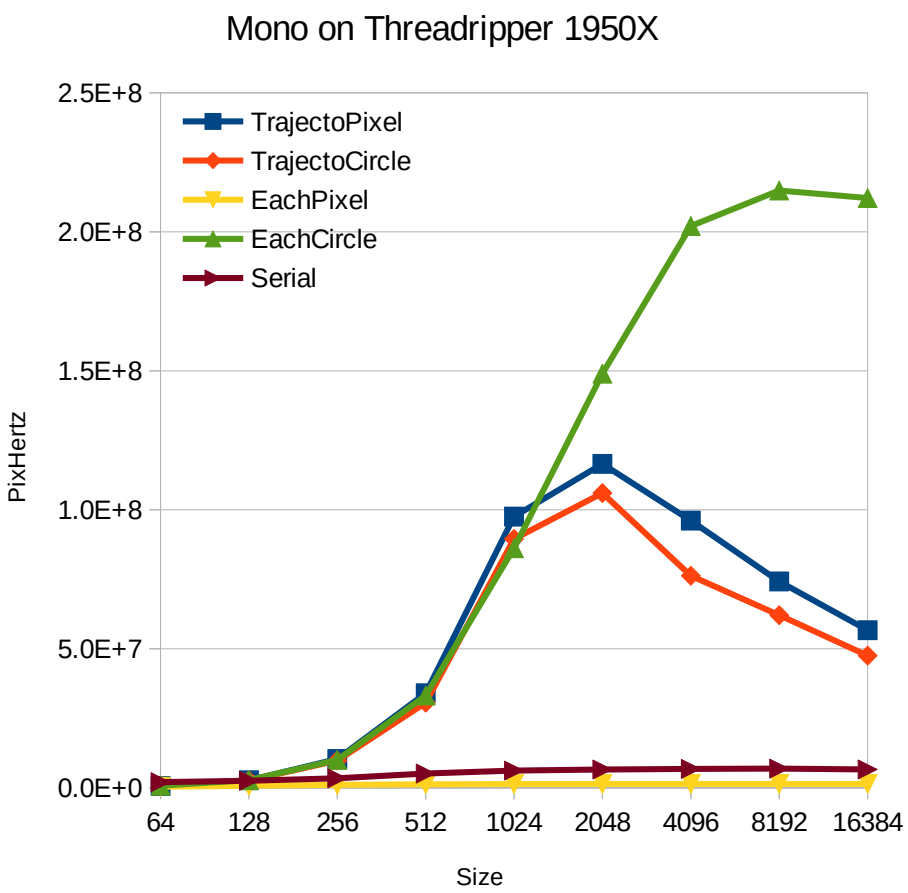
x68 millions in BB and x14 millions in Mono

OpenCL : distribution of computing Which parallel rate PR to use ?

- Initial approach of original C code : **EachCircle**
  - Parallelized along impact parameter : PR=Taille/2
- Brutal approach : **EachPixel**
  - Parallelized along the number of pixels : PR=Taille*Taille
- Hybrid approach : **TrajectoPixel**
  - First parallelized along impact parameters : PR=Taille/2
  - Second parallelize along each pixel (using trajectories) : PR=Taille*Taille
- Approche hybride sauvage : **TrajectoCircle**
  - First parallelized along impact parameters : PR=Taille/2
  - Second parallelized along each angular sector (using trajectories) : PR=4*Taille
- So 4 methods to explore on our CPUs !

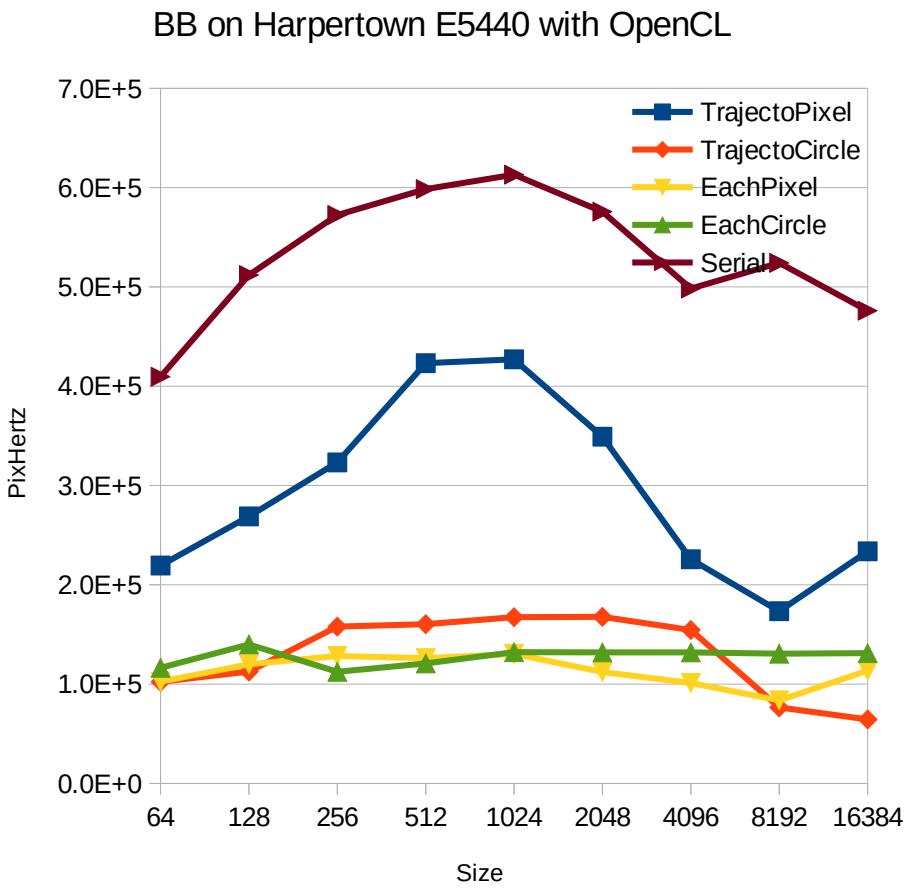
OpenCL on Threadripper 1950x Used method of // very important...

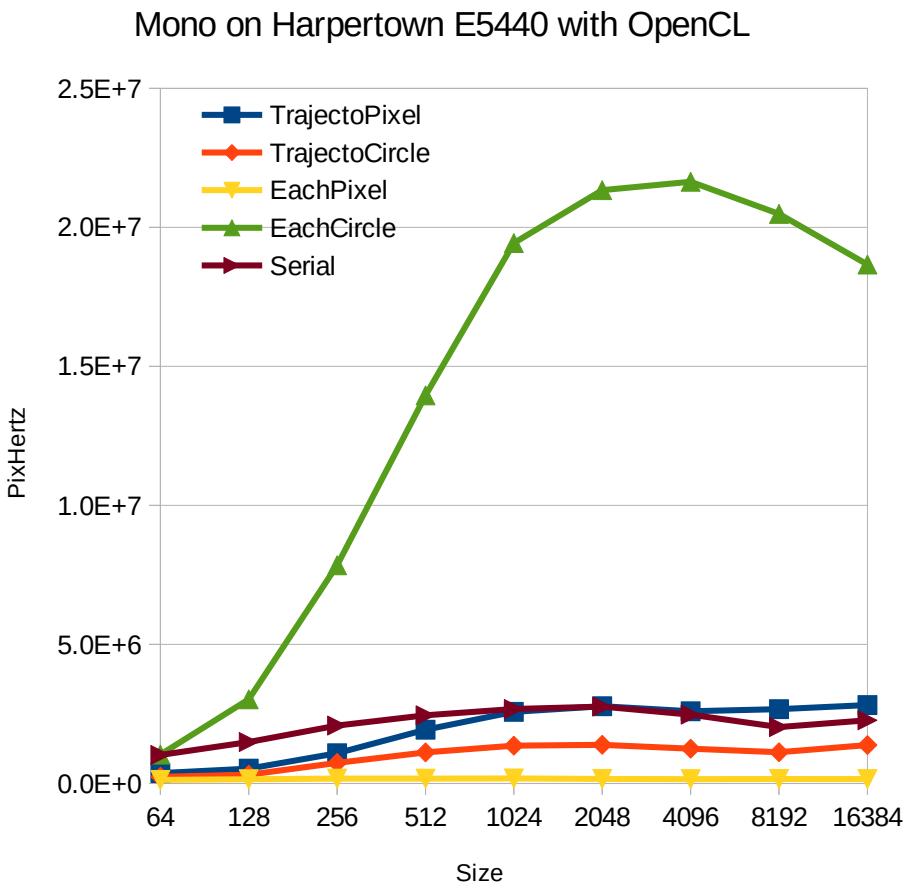




TrajectoPixel for BB, EachCircle for Mono…

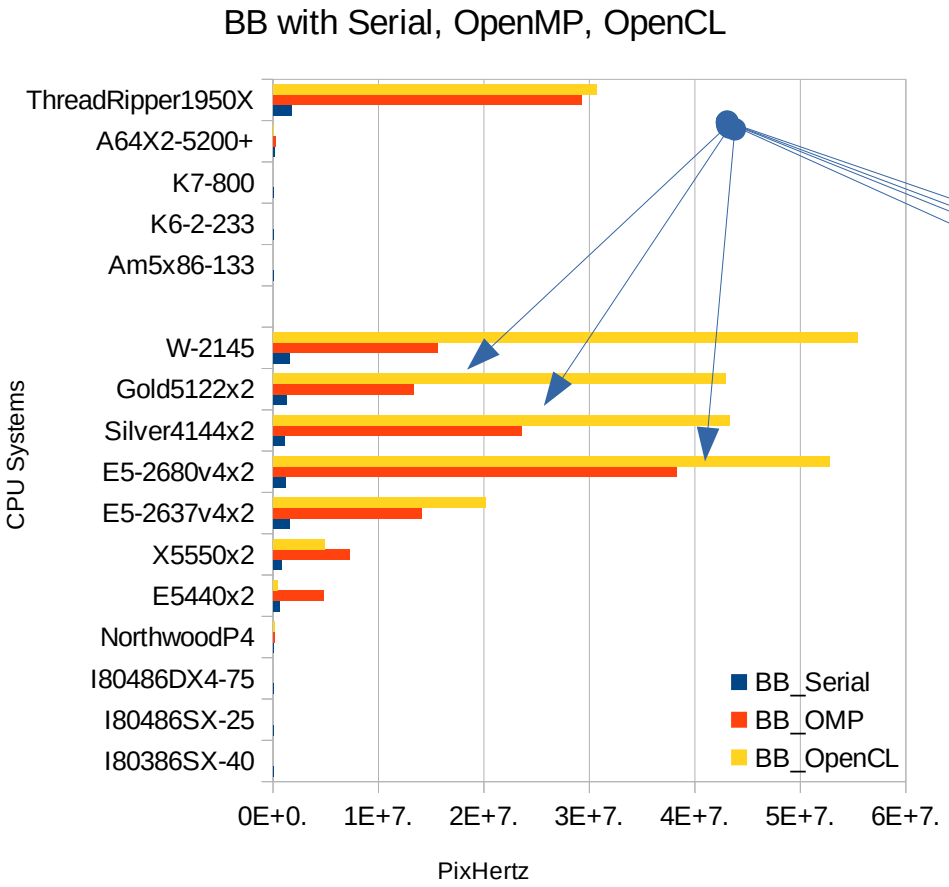
OpenCL on Harpertown E5440x2
OpenCL (AMD) not very efficient

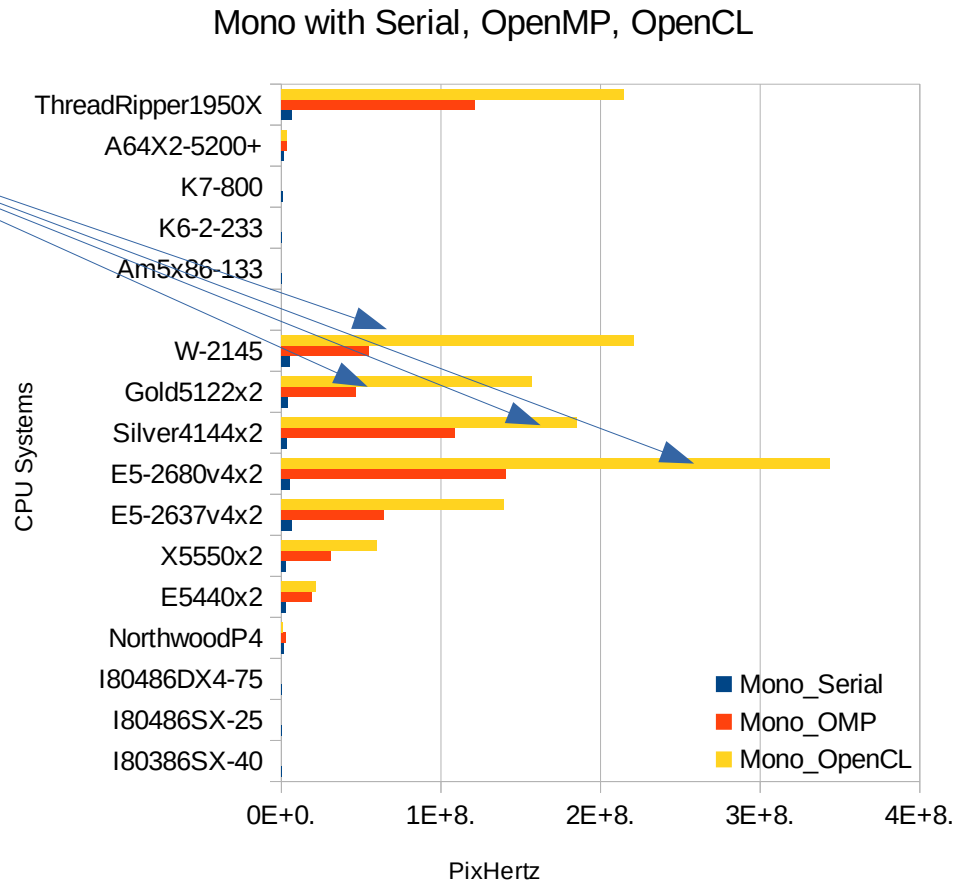




With each CPU, its scalability curve...

OpenCL Parallelization
For all the processors...





OpenCL Intel very efficient on CPU Intel !

Add geatest (GP)GPUs & MIC

Gamer

GPGPU

MIC

AMD GPU

Generations Fermi, Kepler, Maxwell, Pascal, Turing

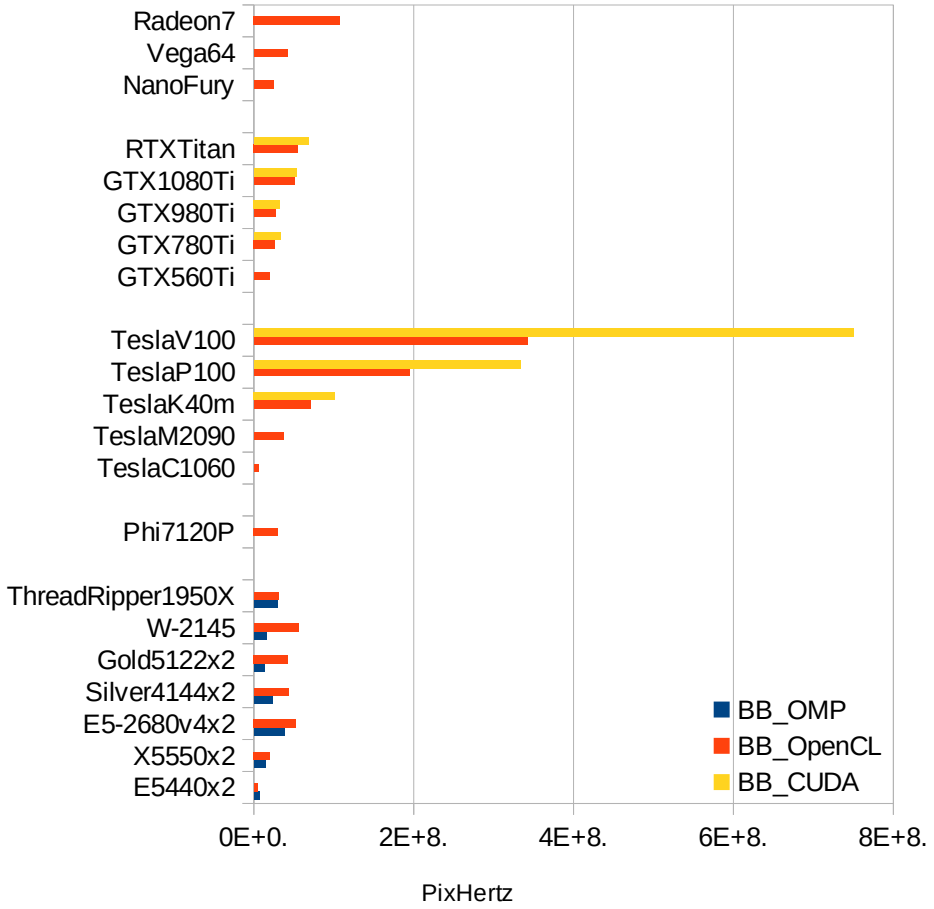
Generations GT200, Fermi, Kepler, Pascal, Volta

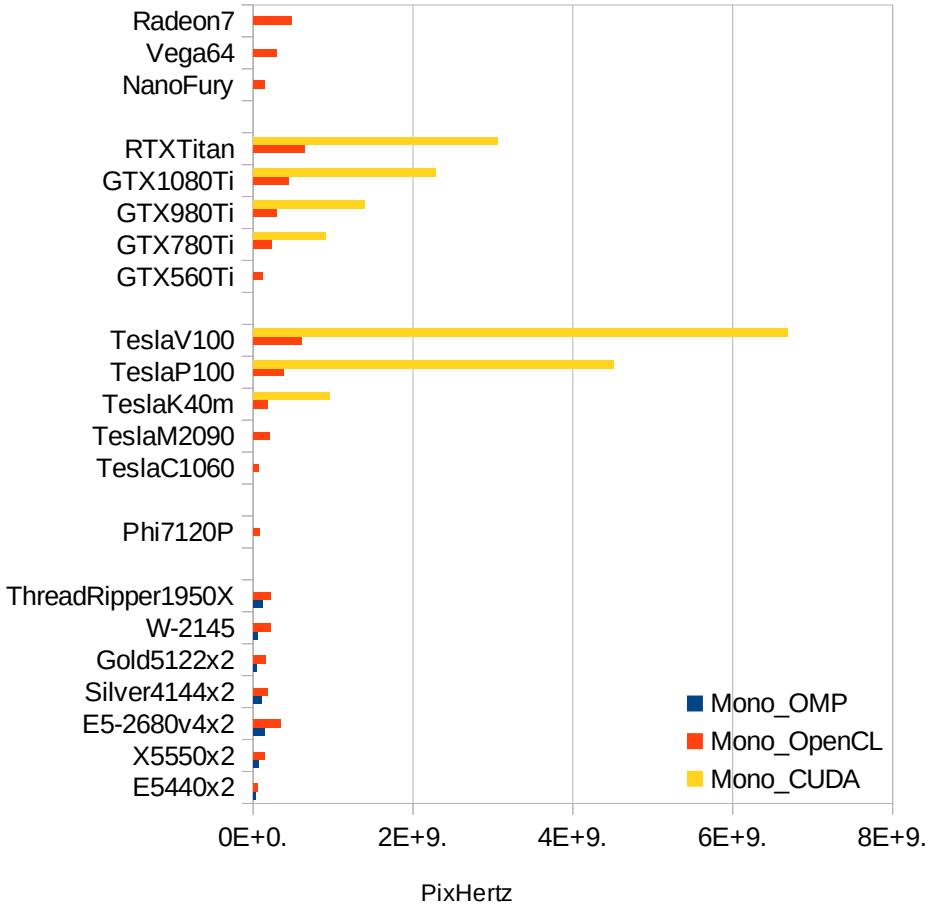
Intel : KNC

R9-Nano, Vega 64, Radeon 7

The bost of computabry of each generation...

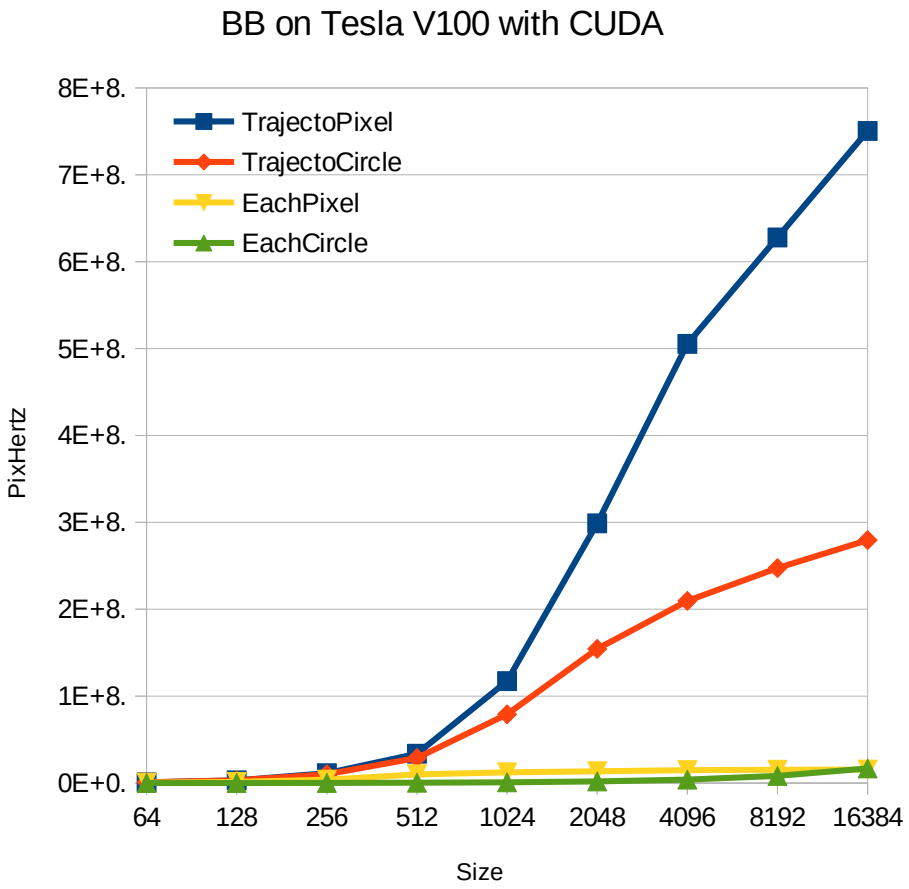
By adding CUDA and GPUs, There is Tesla V100 and the others

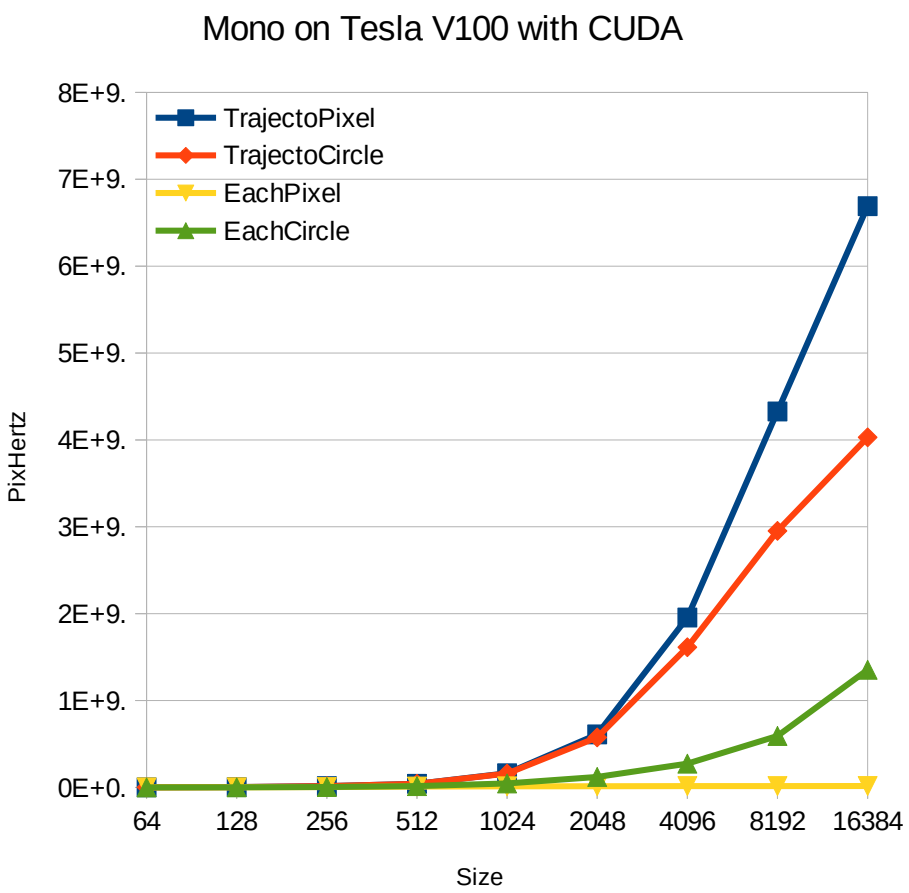




Speedup from **x13** in BB et **x20** in Mono on best CPU.

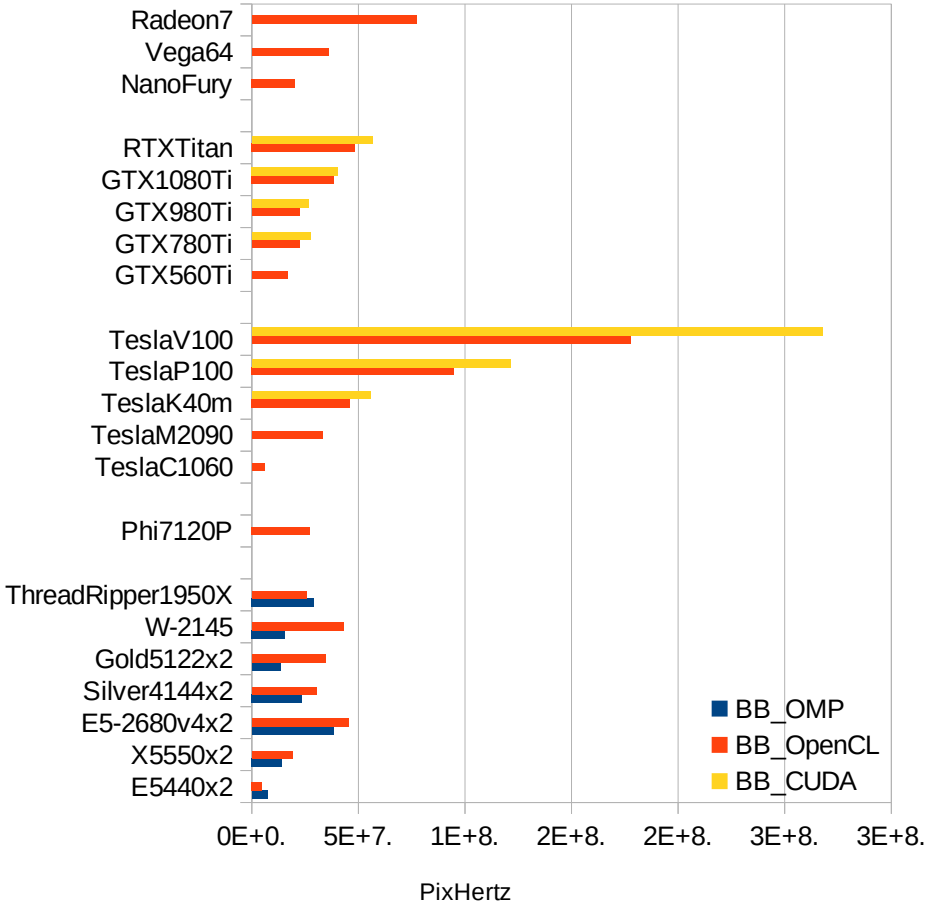
CUDA with Tesla V100
What scalability curves ?

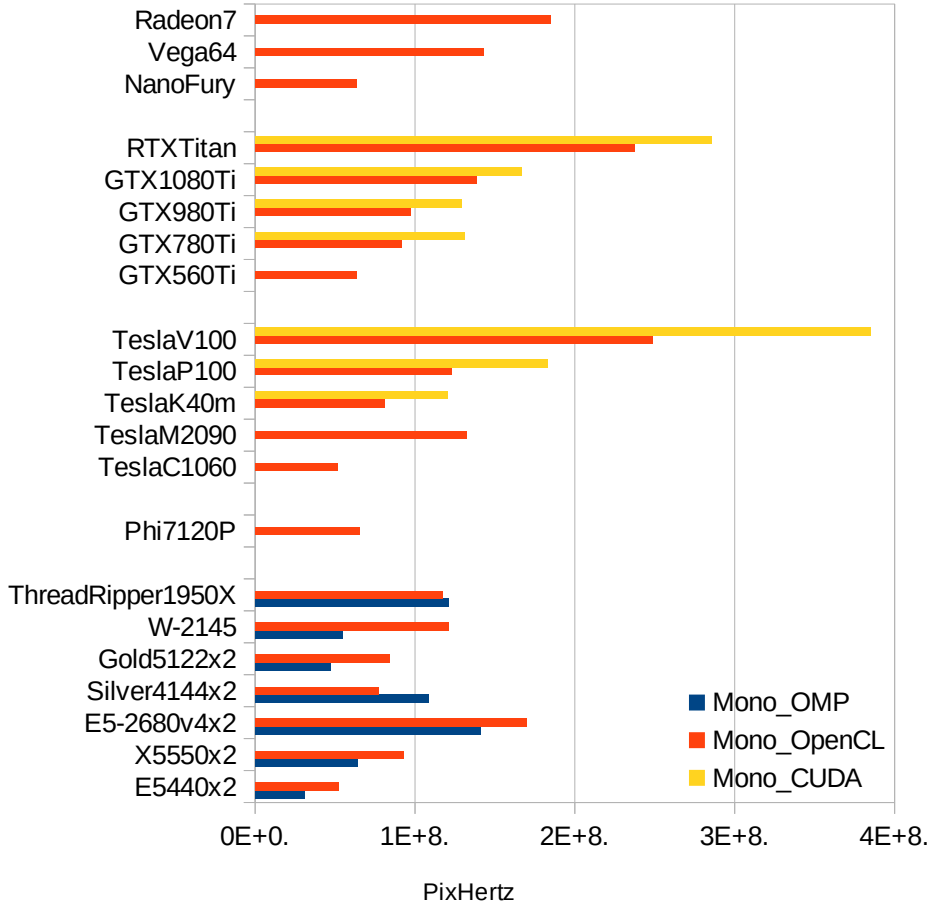




TrajectoPixel definitely the Best !

But I am a big liar (as Nvidia)... Data transfert in CUDA & OpenCL...





On large images, transfert time > compute time

From SAROS to Matrix codes Conclusion

- What about « from scratch » codes :
  - From 7 to 30 speed up for highly parallelized process
- What precautions to take ? Ask the right questions...
  - What is the grain of my code ? Fine or coarse ?
  - Is my kernel computing code small (< hundred of lines) ?
  - Is my parallel rate larger than 100000 ?
  - Can my code fit in 32 GB of RAM ? Stay in RAM ?
  - Is 32 bits computer operations sufficient ?
  - Which SAROS algorithm is near from yours ?
- If yes, you will reach near the best of GPU capabilities, as :
  - OpenMM, Lammps, Hoomd-blue, ...

Not only code & hardware…
Use case is IMPORTANT !