ESDW 2019

Computational Physics

Travels, tries, traces, traps, tricks, trends and trolls...

Benchmarking & Performance (of codes) on myri-ALUs...

Logbook of a native physicist inside the world of parallelism

Emmanuel Quémener

Warnings about your lecturer...

- I'm french
  - And all TV series are translated in France (so, no improving english via TV :-( )
- I'm a « production » of french university 30 years ago
  - And english learning & speaking was not clearly a priority...
- I'm not graduate in computers
  - But I use computers since 1984 and Debian Linux for 24 years...
- I'm a physicist
  - And I work on optical processing for pattern recognition 25 years ago...
- I'm research engineer
  - But I improve my knowledge on all IT domains since 25 years…
- The most important thing I learn this 35 years :
  - « If you can not prove that the work is done, it is not worth undertaking it ! »

Warnings about this course
What it will **not** be ...

- A introduction to general parallel computing
  - https://computing.llnl.gov/tutorials/parallel_comp/
- An introduction to parallel langages
  - MPI : https://computing.llnl.gov/tutorials/mpi/
  - Posix Threads : https://computing.llnl.gov/tutorials/pthreads/
  - OpenMP : https://computing.llnl.gov/tutorials/openMP/
  - **That's where I learn alone how to…**
- But I would like to share what's never told !

Codes, Performance & Benchmark : Definitions : « return to the source »

- Etymology (Etymonline) & Definitions :
  - **Code** : from latin codex « book, book of laws »
    - « systematic compilation of laws » (1236)
    - « system of telegraphic communication » (1866)
  - **Performance** :
    - « accomplishment » (of something)
    - meaning « a thing performed » is from 1590s
    - « set of optimal capabilities for a system » (1929)
  - **Benchmark** :
    - From bench-mark, « surveyor's point of reference » (1838), figurative sense since 1884 :
      - « a standardized problem or test that serves as a basis for evaluation or comparison »
- And we will choose for our studies...
  - **Code** : both :-), **Performance** : three of them :-), **Benchmark** : the figurative sense...

If computing was cooking…
Code : only the recipie...

Code ~	Recipie
Computer ~	Kitchen
Input Data ~	Ingredients
Output Data ~	Meal Dish
Process ~	Cooking process
Control Unit ~	Cooker
ALU ~	Utensil
Me ~	Client
Batch Request ~	Order

Some definitions and letters...

- ALU : Arithmetic & Logic Unit
- CPU : Central Processing Unit
- Flops : Floating Point Operations Per Second
- (GP)GPU : (General Purpose) Graphical Processing Unit
- MPI : Message Passing Interface (communication between nodes)
- RAM : Random Access Memory
- SMP : Shared Memory Processors
- TDP : Thermal Design Power
- And several new ones :
  - PR : Parallel Rate (NP in MPI, Threads in OpenMP, Blocks, WorkItems in GPU)
  - Itops : Iterative Operations Per Second
  - EPU : Equivalent Processing Unit (optimal parallel rate deduced)

What's this ?
Code, protocol of experimentation

- In cuisine :
  - We have all the ingredients, we want to make a dish !
- In scientific ways :
  - Simulation : « On Its Theory (Discrete ?) Service »
  - Processing : for « demanding » experimenters
  - Visualisation : to see to perceive things (and share)
- Each launch is an experience (and unique one)...
  - Recipies : « codes » becaming « workflows »
  - Utensils : librairies, OS, hardware, networks, ...
  - Ingredients : modelisation, data, ...
  - Execution : and the experience cannot be reduced only to Results

Families of Codes

- What distingish the different codes I use ?
  - « My code I did myself and I'm proud of »
  - My supervisor code
    - In fact, the stratification of codes produced by previous generations of students
  - Code «business»
    - « Ikea » model : delivered with assembly instructions (without toolbox)
    - « Crozatier » model : (almost) ready to use
- Dependencies to :
  - Generic librairies : BLAS, Lapack, FFTw
  - Proprietary librairies : Mathworks, Intel, Nvidia, AMD, …
  - Hardware !

Performance : how ?
A question of objectives !

- To put all luggages & family inside the car
- To draw the attention of females outside the night clubs
- To get from point A to point B in a town with traffic jam
- To climb to Pikes Peak




Performance : how ?
A question of observables !

Sport performance

- To run a 100-meters ?
- To run a marathon ?
- To make shot put ?
- To complete an heptathlon ?

Performance : Conditioned by objectives

- Speed : elapsed time (only?)
- Work : immobilization of resources
- Efficiency : best use of available resources
- Scalability : incremental progress when more resources are dedicated
- Portability : diffusion to other IT infrastructure
- Maintainability : time spent to maintain the system operational
- General approach :
  - Define un criterium
  - Research extreme values (maximum or minimum) for a pertinent test suite

Speed as Performance Criterion « Speed, I'm Speed... »

- All time, but not only « Elapsed time »
- To use code : the 3 costs
  - Entry cost : to learn, to write software, to integrate inside infrastructure, …
  - Operational cost : to maintain, to operate
  - Exit cost : substitution by an equivalent code, an equivalent technology (Cell...)
- Optimization (and its problem) : DD/DE > 1 is relevant ?
  - DE : Total elapsed time for my code
  - DD : time spent to minimize this total elapsed time
- To estimate the value :
  - System tools, metrology tools in langages, codes, ...
  - « Et après moi ? Le déluge ? » : what future for the code ?

Work as Performance Criterion

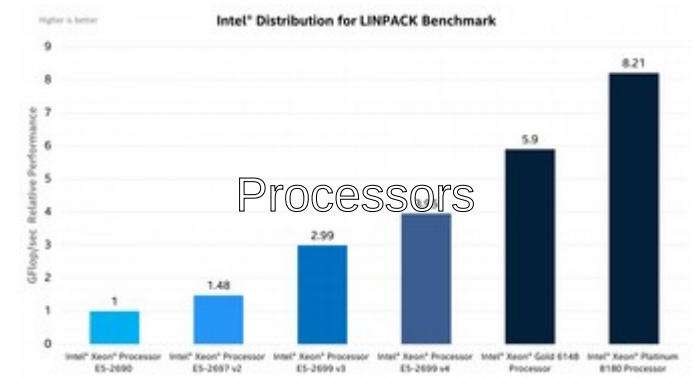
- Work : « Time is money »
  - Resources : CPU, RAM, GPU, storage, network, ...
  - In fact, a Matriochka :
    - CPU : several cores, CU, ALU, piles, ...
    - RAM/SRAM : 4 levels
    - Storages : local, slow & shared (NFS), fast & shared (GlusterFS, Lustre, ...)
    - Networks : slow (Gigabit), fast & low latency (InfiniBand)
- Job : reservation (& immobilization) of resources
  - Classical : Nodes * Elapsed time
- For a code, « system fingerprint »
  - Profiling tools, System tools

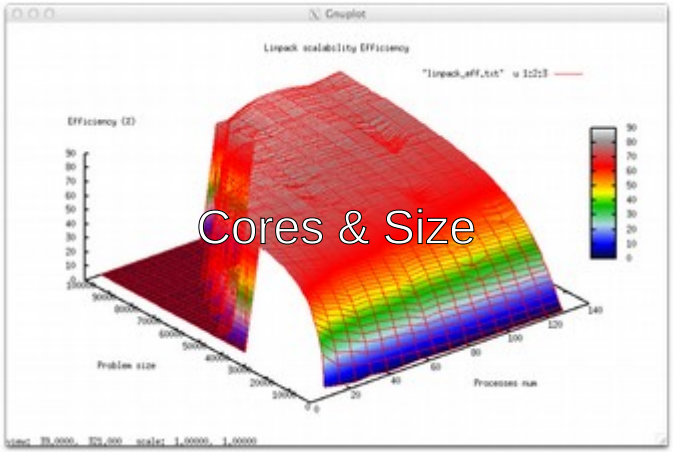
Scalability as Criterion

- Scaling :
  - In the tasks to be done : Elapsed time ? f(Elapsed Time)
  - In required resources : g(System Resources)
- Reefs to avoid :
  - Scaling effects (in fact, threshold effects are even worse)
  - Needing conductor ? From a Quatuor to a symphony orchestra…
  - Although you execute, the available resources are limited…
    - You think I'm joking :-/ ?
  - **Parallelization becomes unescapable, but why ?**

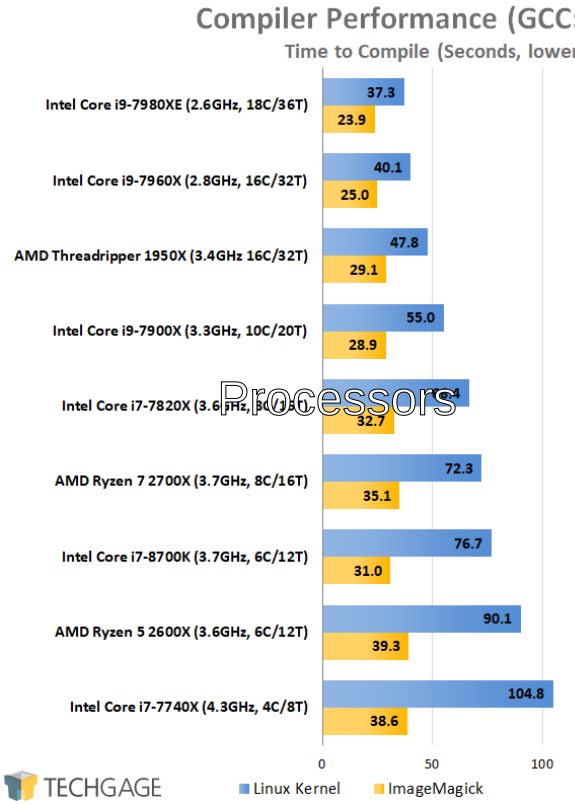
Performance of Computers Between Linpack & Phoronix...

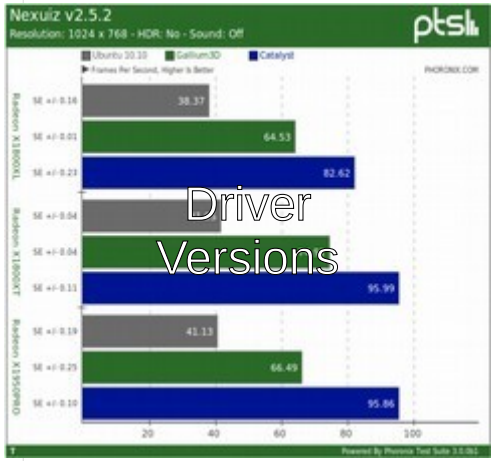
Linpack

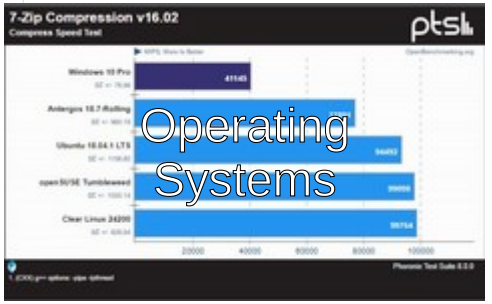




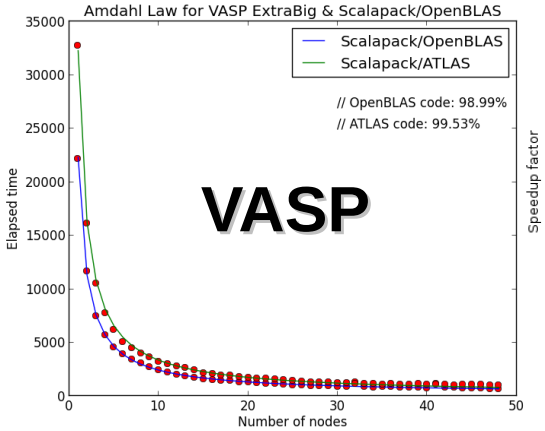
Phoronix

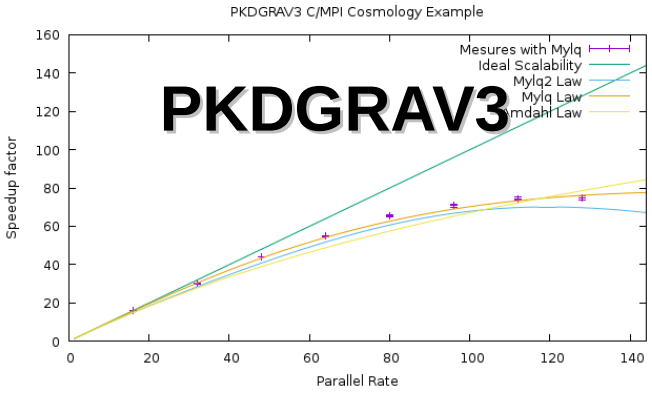


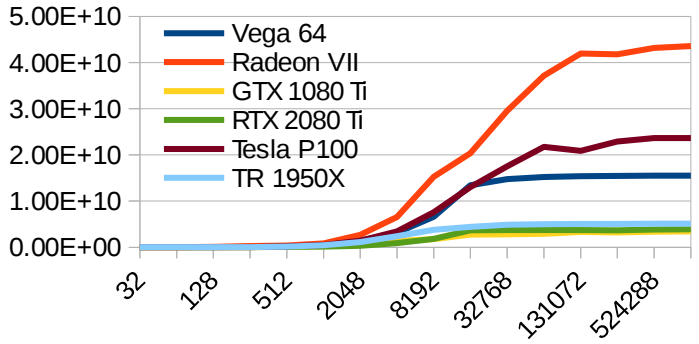


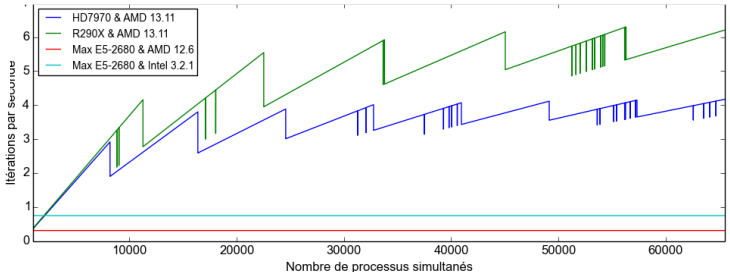


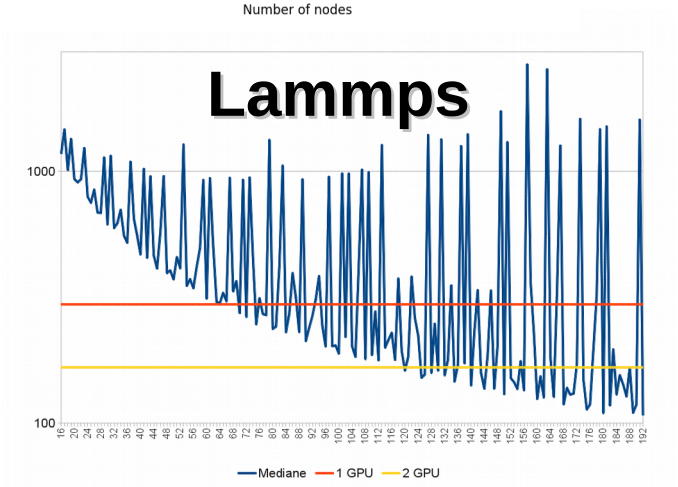
« Business » codes & hardware
First steps in scalability

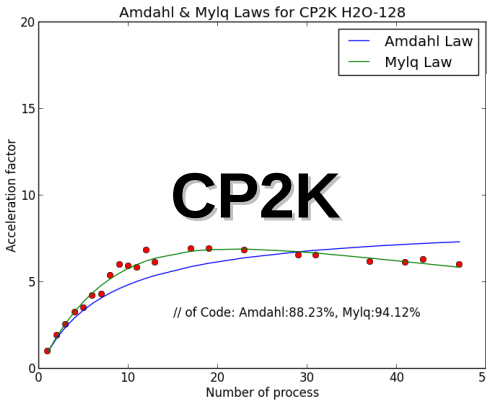


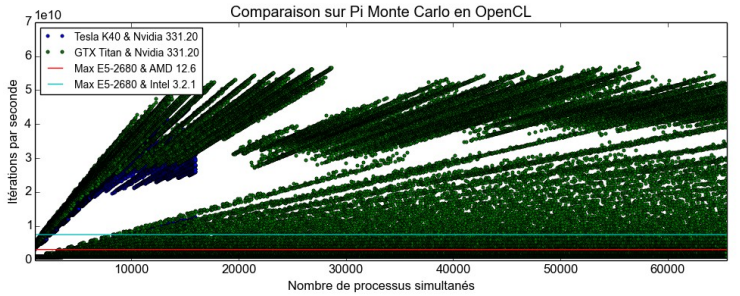






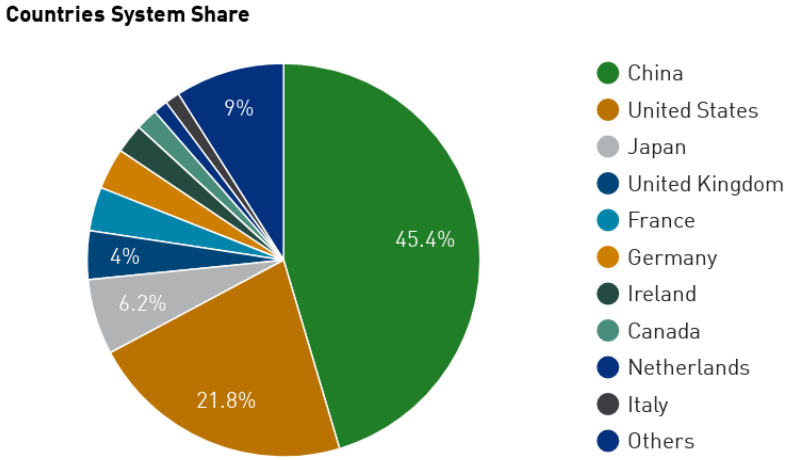


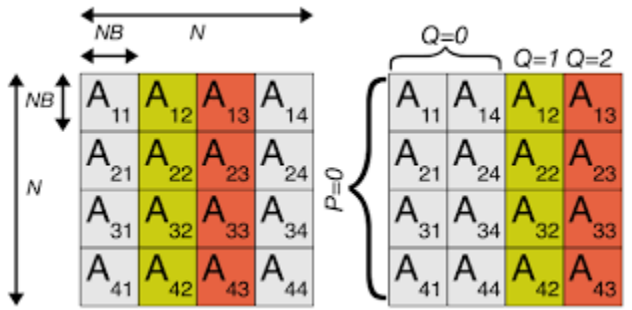




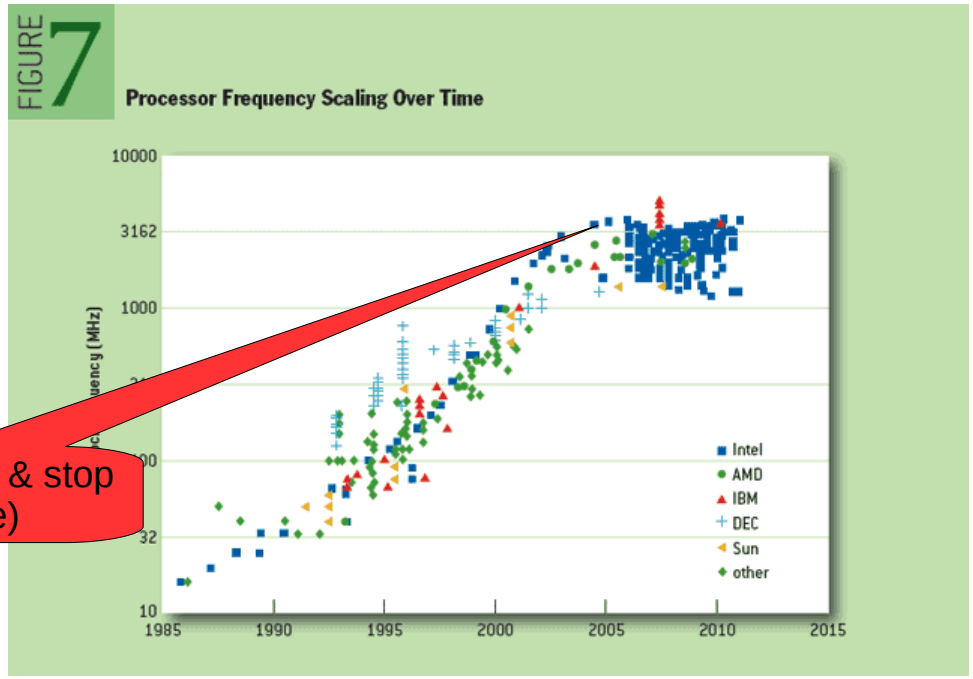
Performance of Computers Linpack & the Top 500

- **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...





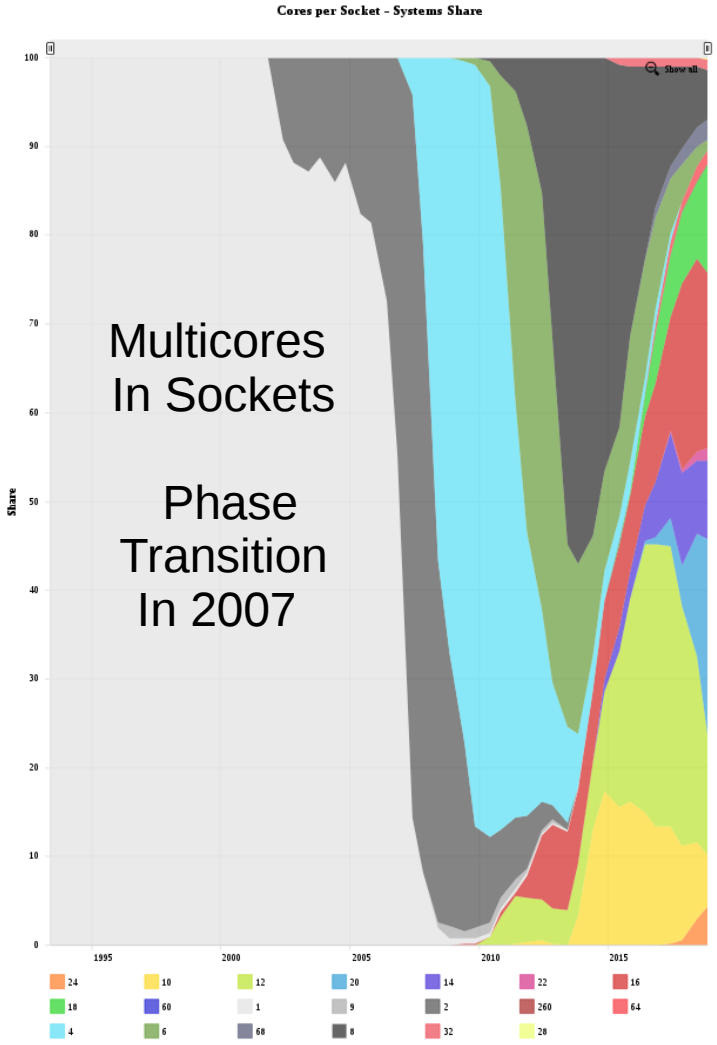
Linpack & the Top 500
The Moore Law respected… How ?

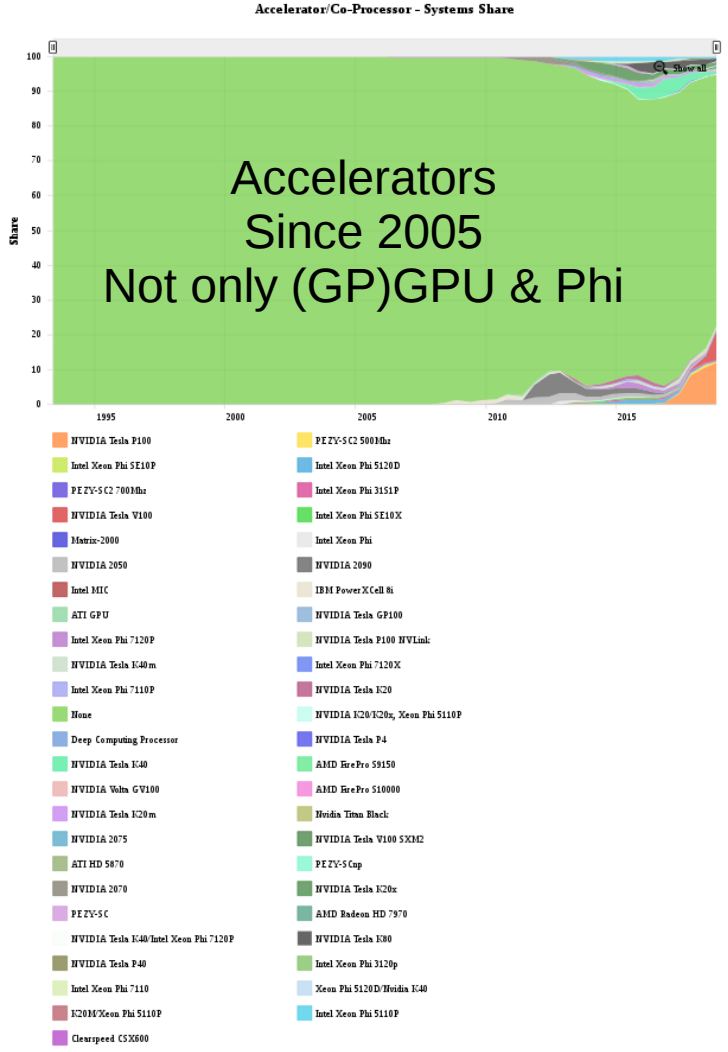


x1000 in 20 years & stop (or decrease)

- 2 ways to break heat-wall
  - From multi-cores to many-cores
  - Accelerators with Myri-ALUs

Linpack & the Top 500
The Moore Law respected… How ?





Linpack & the Top 500
How to retreive Rpeak :-/ ?

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	200,794.9	9,783
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371

- Cores : processing units
- Rmax : result of HPL
- Rpeak : theoritical perf'
- Ratio max/peak ~ 73 %

- The China one, the third...
  - Cores = 40960 SW26010
    - 260 cores/SW26010
  - Rpeak ~ Cores*1.45e9*8
  - So : 8 instructions / cycle

- The US one, the first...
  - Cores = 9216 Power9 + 27648 V100
    - 22 cores/Power9, 80 StreamMuliprocessors/V100
  - Rpeak ~ $Rpeak_{CPU}$+$Rpeak_{GPU}$
    - $Rpeak_{CPU}$ = 9216*22*3.07e9*64 ~ 20 %
    - $Rpeak_{GPU}$ = 27648*80*1.13e9*64 ~ 80 %
  - So 64 instructions / cycle on CPU&GPU

So much « Instruction Per Cycle » ! How it's possible ?

- Deeply & old evolution inside CPU :
  - RISC replaces CISC :
    - RISC : 1 instruction per cycle
  - Pipelining becomes the « rule » :
    - The 5 operations are executed in 1 cycle
  - Floating Point Unit is integrated inside CPU
    - It as a specific ALU
  - Multiplication of specialized ALU inside CPU
    - From 5 in AMD K6 to 10 per core on Zen architecture
  - ALUs integrate vectorization :
    - Firstly for 3D operations (MMX, 3DNow, SSE, SSE2, …, AVX512)
  - Socket integrates several cores (UC+ALU+Cache memories)

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Linpack & the Top 500
The power of Hybrid machines

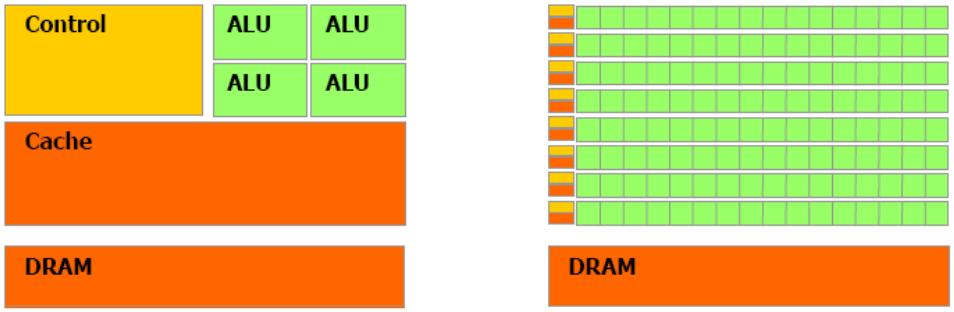
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	200,794.9	9,783
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384
6	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,072	20,158.7	41,461.2	7,578
7	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	32,576.6	1,649
8	**SuperMUC-NG** - ThinkSystem SD530, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path , Lenovo Leibniz Rechenzentrum Germany	305,856	19,476.6	26,873.9	
9	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
10	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

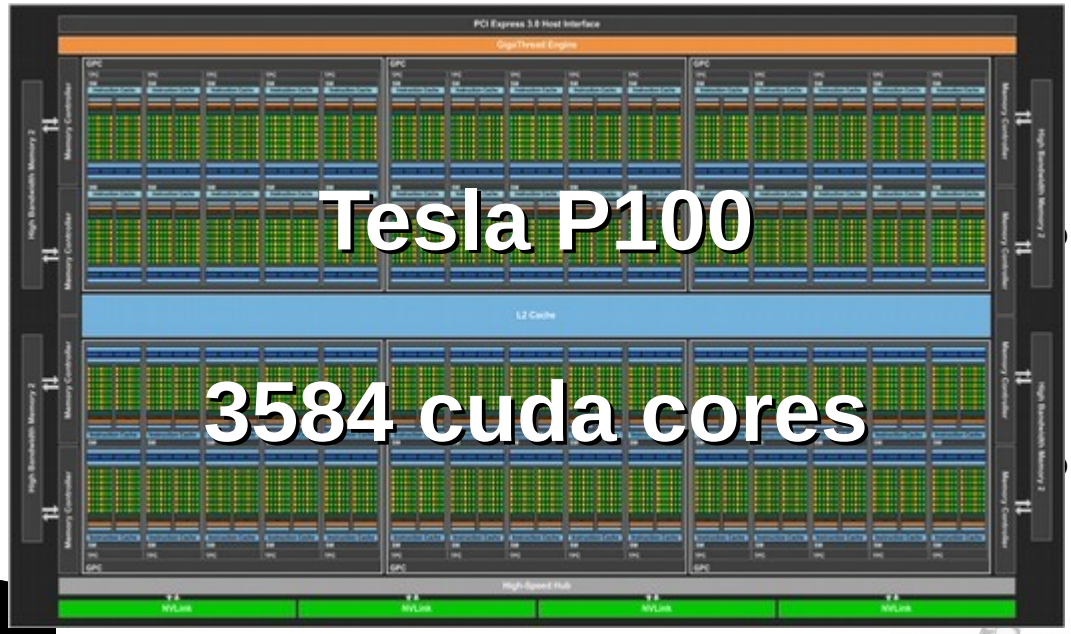
With (GP)GPU

With Accelerator Xeon Phi

- November 2018 :
  - 7/10 Hybrid on the Top 10
  - 25 % of the Top 500
- Before 2012 :
  - Hybrid : MPI+OpenMP
- After 2012 :
  - OpenMP+Cuda
  - OpenMP+MPI
  - OpenMP+Cuda+MPI
  - OpenCL+MPI

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

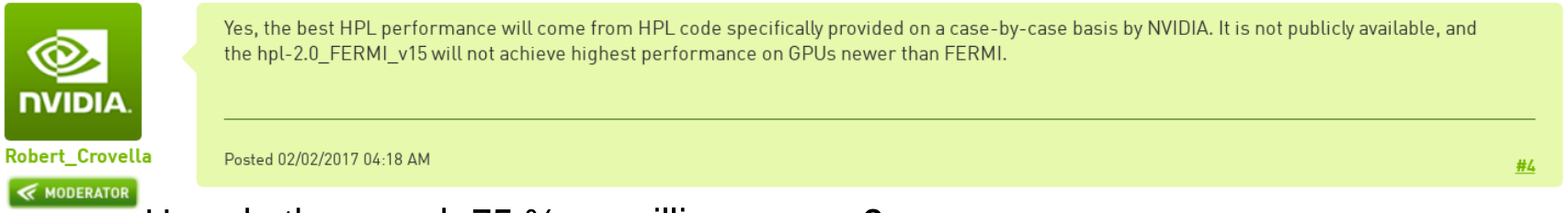




- Operations
  - Matrix multiplications
  - Vectorization
  - « Pipelining »
  - Shader (multi)processeur
- Programming : 1993
  - OpenGL, Glide, Direct3D, ...
- Genericity : 2002
  - CgToolkit, CUDA, OpenCL

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)



  - 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**

What's Blaise Pascal Center ?

Director : Pr Ralf Everaers

- Centre Blaise Pascal: « maison de la modélisation »
  - Hotel for projects, conferences, trainings on all IT services
- Hotel for projects:
  - Technical benchs in an experimental platform for everybody…
  - Digital laboratory benchs for laboratories for specific purposes
- Hotel for trainings:
  - ATOSIM (Erasmus Mundus)
  - Continuing educations for searchers, teachers & engineers
  - Advanced education : M1, M2 in physics, chemistry, geology, biology, …
- Test center : to reuse, to divert, to explore in HPC & HPDA

Multinodes TechBenchs : 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

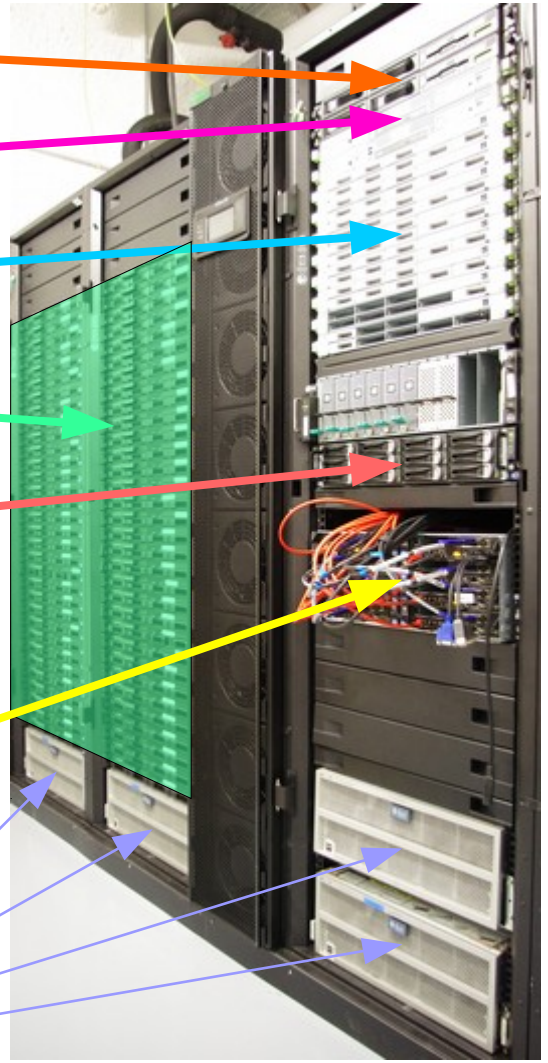
4 nodes Dell C6100 with Xeon X5650
48 physical cores HT @2666MHz
Interconnection Infiniband QDR 40 Gb/s
+ C410X with 4 GPGPU

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

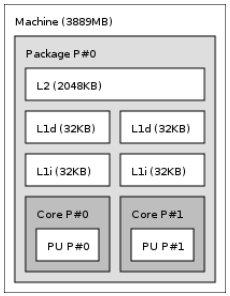
8 nodes HP SL230 with Xeon E5-2667
64 physical cores HT @2666MHz
Interconnection Infiniband FDR 56 Gb/s

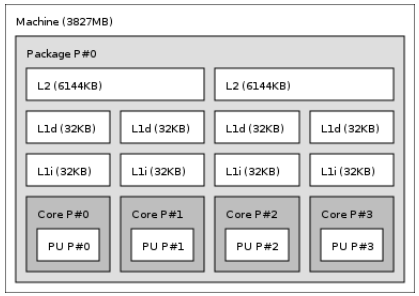
8 nodes Dell R410 with Xeon X5550
64 physical cores HT @2666MHz
Interconnection Infiniband DDR 20 Gb/s

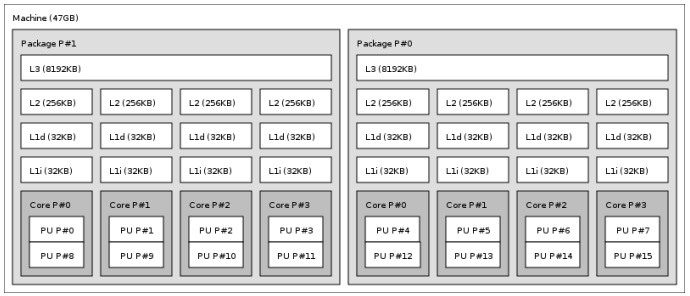
4 nodes Sun X4500 with AMD 285
16 physical cores @2400MHz
Interconnection Infiniband SDR 10 Gb/s

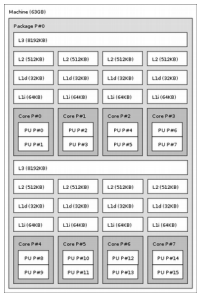


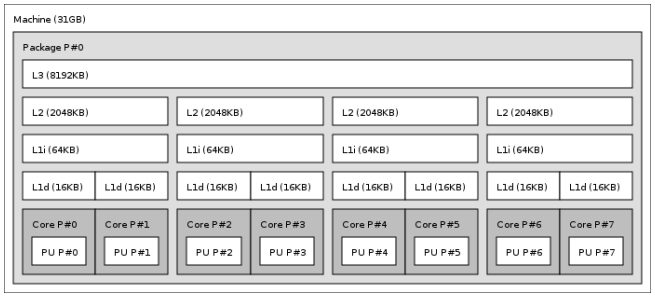
Multicores TechBenchs
From 2 to 28 cores: examples...

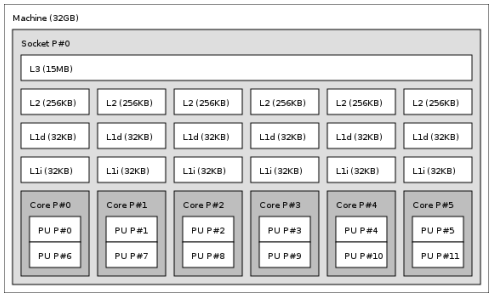


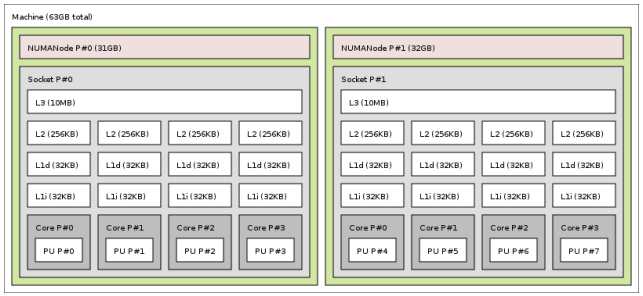


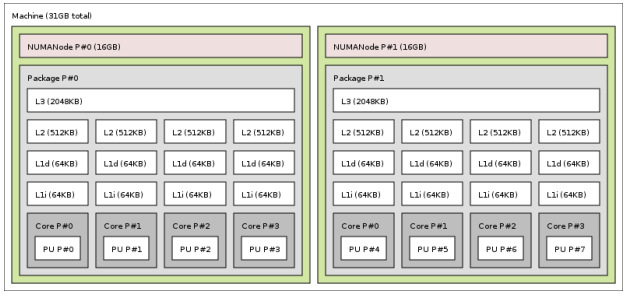


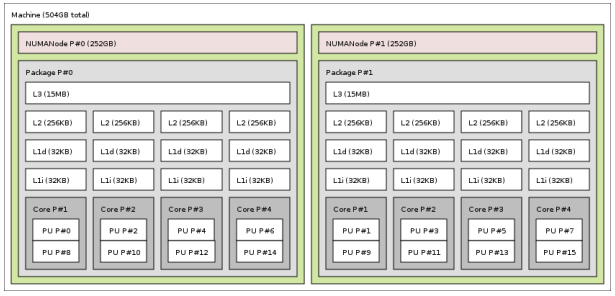












Multishaders TechBenchs (GP)GPU

77 different models...

GPU Gamer : 21

- 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 1660 Ti~~

GPGPU : 9

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

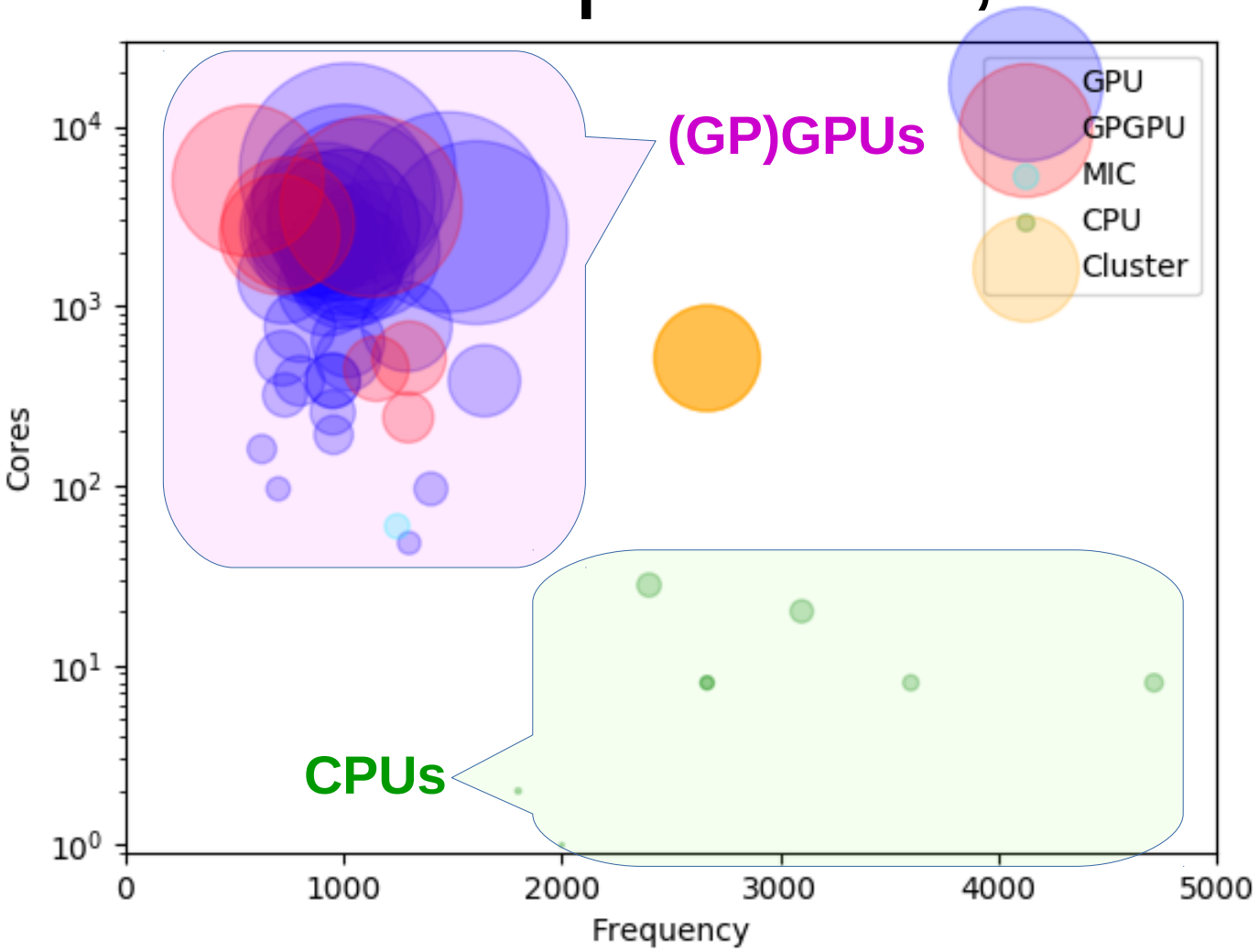
GPU desktop & pro : 28

- ~~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 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~~

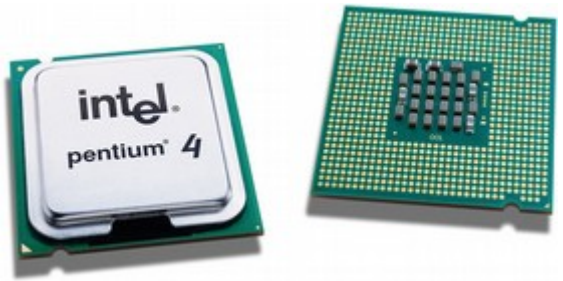
GPU AMD Gamer & al : 19

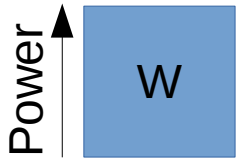
- ~~HD 4350~~
- ~~HD 4890~~
- ~~HD 5850~~
- ~~HD 5870~~
- ~~HD 6450~~
- ~~HD 6670~~
- ~~Fusion E2-1800 GPU~~
- HD 7970
- FirePro V5900
- ~~FirePro W5000~~
- Kaveri A10-7850K GPU
- ~~R7 240~~
- R9 290
- R9 295X2
- ~~Nano Fury~~
- R9 Fury
- R9 380
- RX Vega64
- ~~Radeon VII~~

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



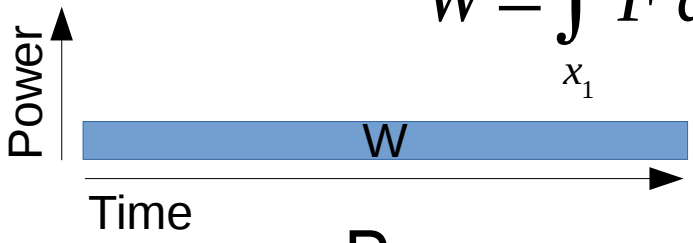
Work on Computing Resources from a Physicist Point of View



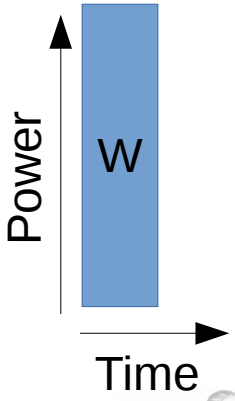


Mechanics

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



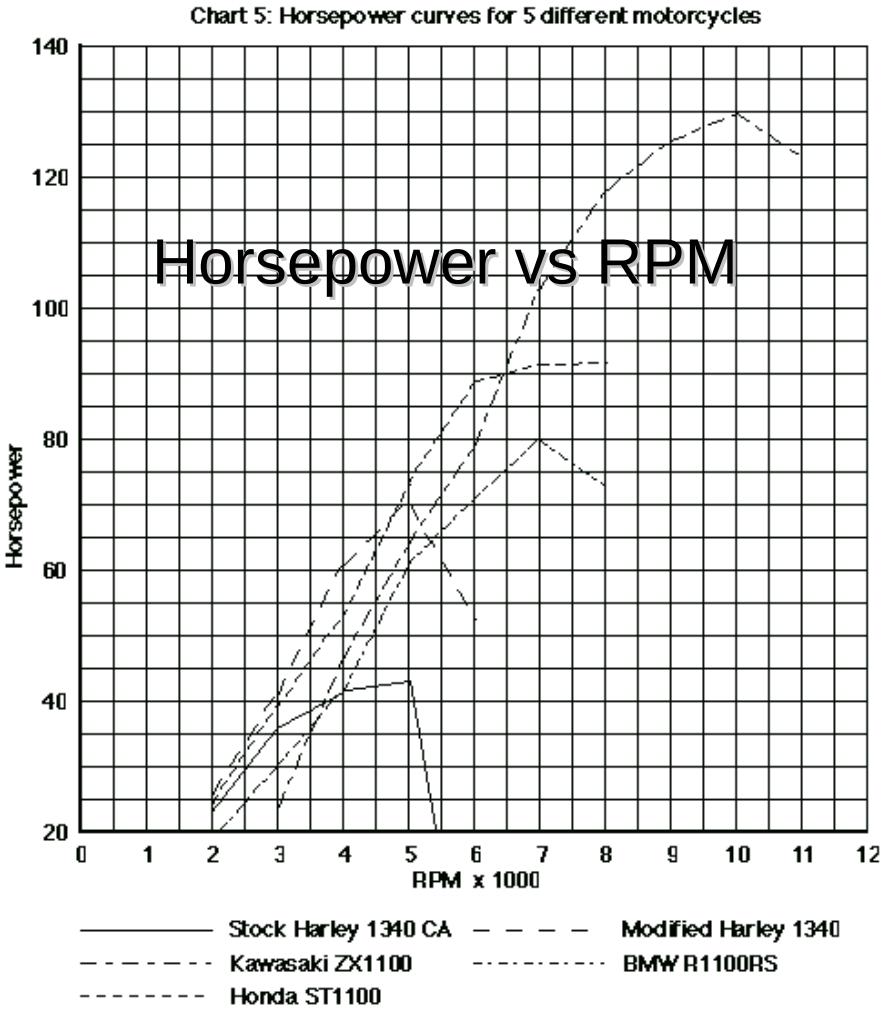
Thermodynamics

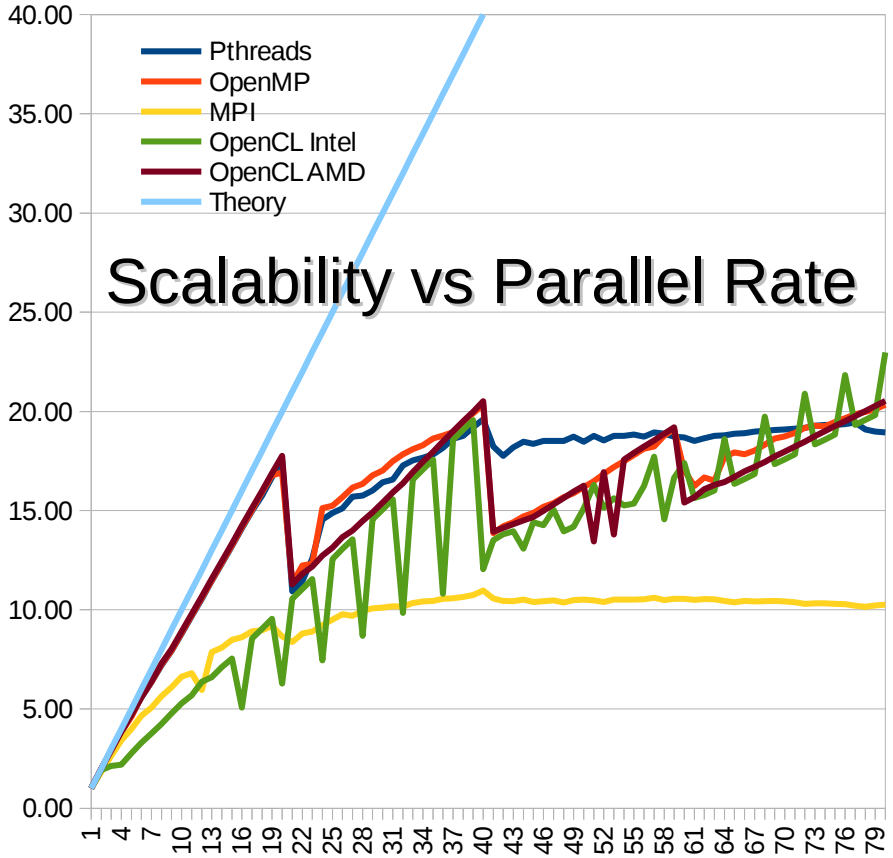


Power equals product :

- Frequency
- Number of Workers
- Power of 1 Worker

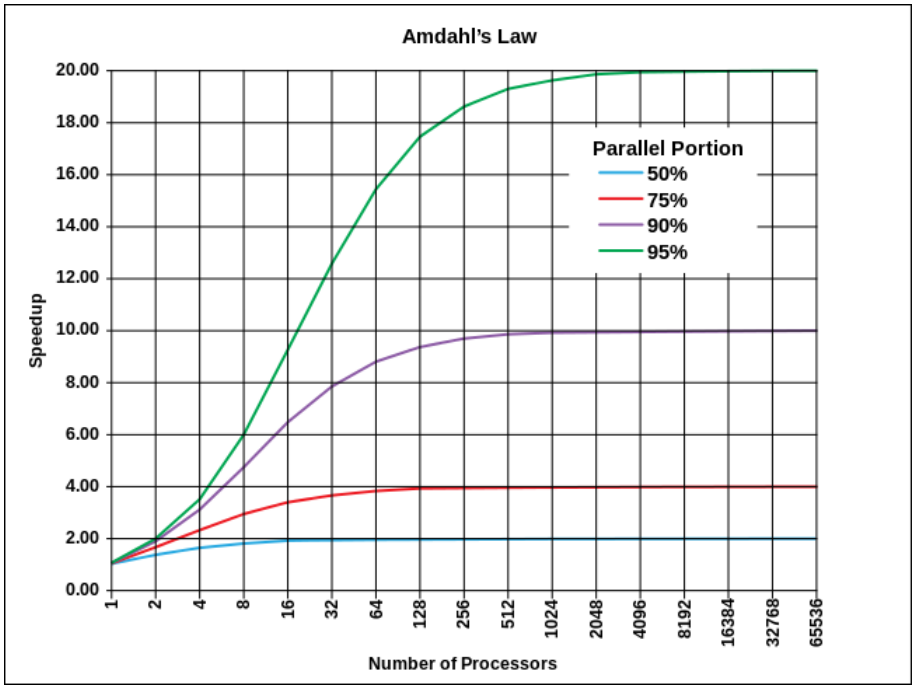
Work on Computing Resources
An Engine as the source of Power





How to estimate // Efficiency ? Amdahl Law, order (and decay)

- In the process, 2 parts
  - Sequential part, in fraction s
  - Parallel part, in fraction p
  - Elapsed Time : $T_N=T_1(s+p/N)$
  - Speedup : 1/(1-p+p/N)
  - Efficiency : 1/N(1-p+p/N)
- Speed up (& efficiency) :
  - 2 systems : N=500 & N=1000
  - 4 cases : 90 %, 99 %, 99.9 %, 99.99 %



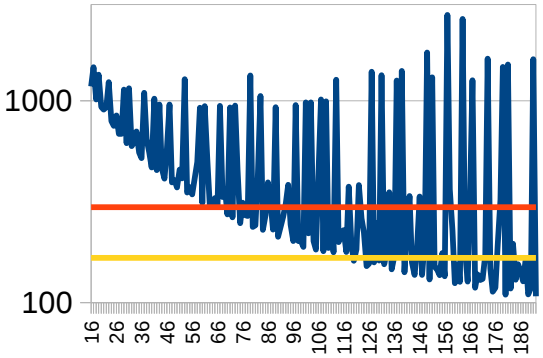
How to estimate Parallel Efficiency ? Amdahl Law, order (and decay)

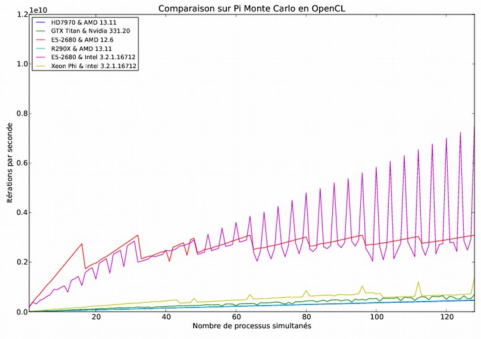
- Speed up (& efficiency) : N=500 & N=1000

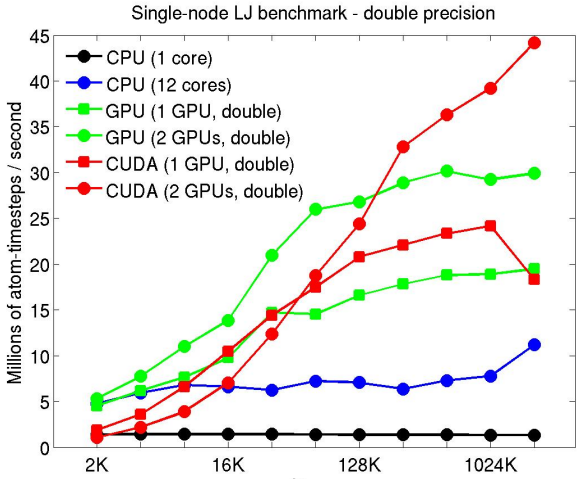
Parallel Rate	N=500		N=1000	
Parallel Part	Speedup	Efficiency	Speedup	Efficiency
90%	9.8	2%	9.9 (+0.1%)	1%
99%	83	17%	91 (+9%)	9%
99.9%	334	66%	500 (+50%)	50%
99.99%	476	95%	909 (+91%)	91%

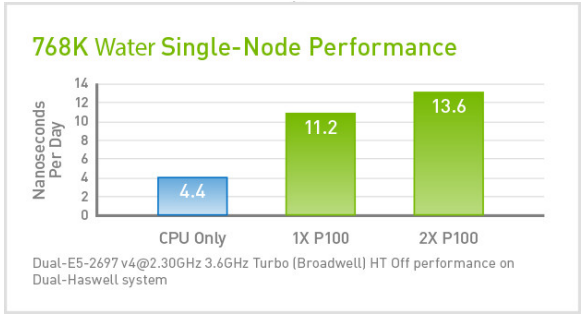
- Questions :
  - What's about scalability of my code ?
  - Is Amdahl law representative of « real » applications ?

GPU as accelerator : Truth or lie ?
Ready to take the « red pill » ?





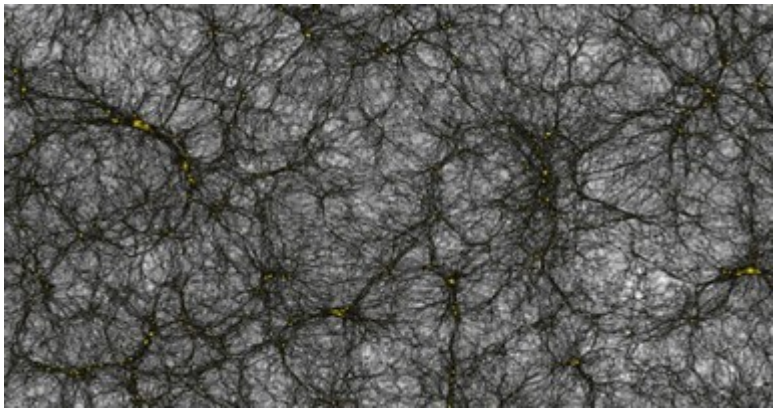




Framework established… What codes to use ?

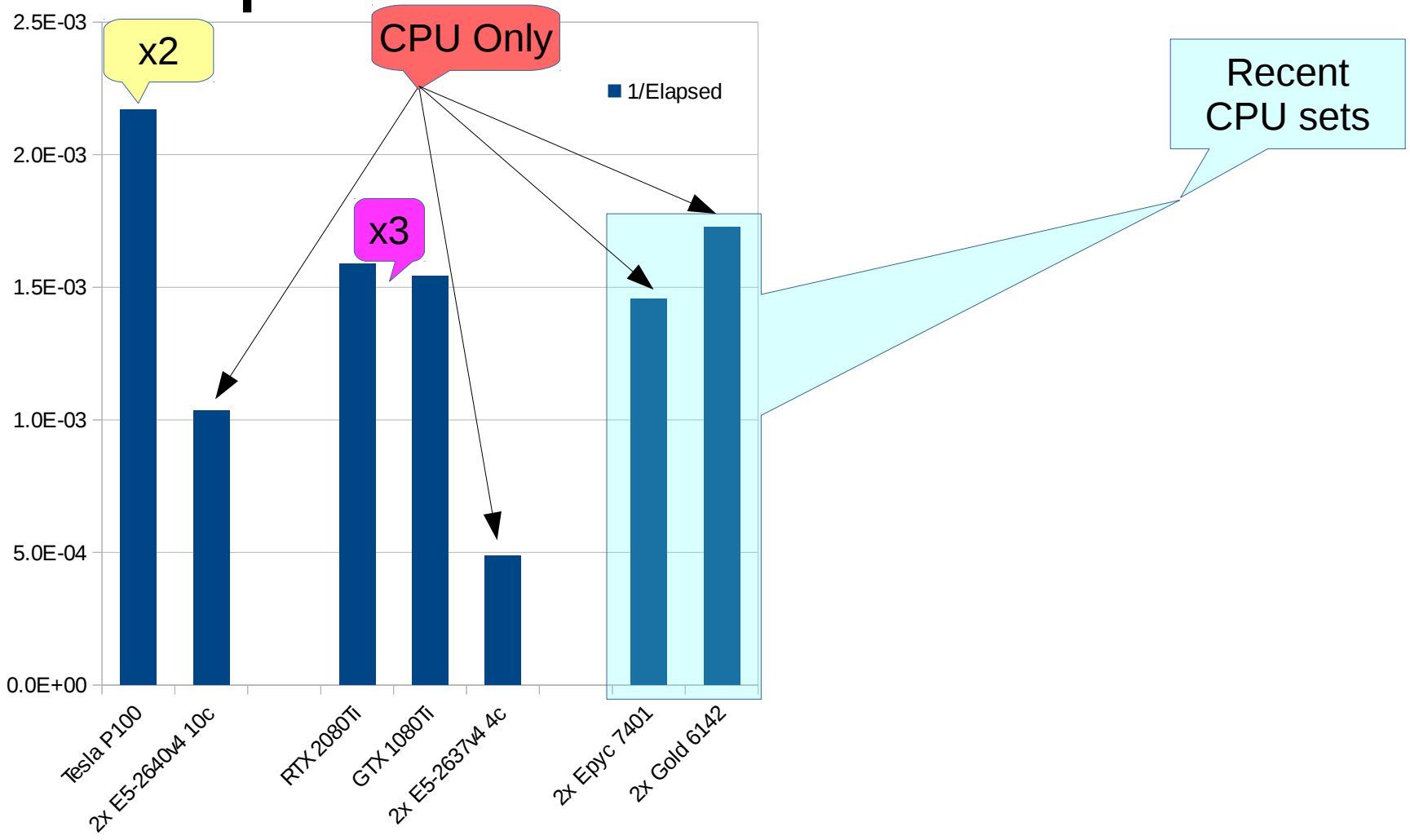
- Code « Ikea » : to compile itself
  - PKDGRAV3 : astrophysic one...
  - Gromacs : molecular dynamic one
- Code « with dependencies » :
  - «integrator» approach :
    - BLAS with xGEMM
  - « developper » approach :
    - Pi Dart Dash, Nbody, Splutter...

PKDGRAV3 : a good customer … and a high exposure on media !



- Characteristics :
  - Open Source
  - Hybrid : MPI, OpenMP, Cuda
- An « easy » compilation
  - Inside the doc (README) :
    - Quickstart : ./configure ; make
    - For CUDA support :
      - ./configure --with-fftw --enable-integer-positions –with-cuda
      - make -j 16

PKDGRAV3
Comparison between CPU & GPU



Gromacs : The good « client » of Chemistry

- Hardware set : technical benchs of CBP
  - 76 different GPU, 22 models of CPU
- Software set : Debian 9.4 « stretch » on AMD64
- Specifications of software :
  - Open Source
  - Hybride : multi-nodes, multi-cores, multi-shaders
  - Large execution properties CPU (MPI+OpenMP), CPU+GPU
  - Easy parallel tuning and GPU choosing

Gromacs : use case Nvidia Launch parameters

- Time-consuming task (& parallelized) : mdrun
  - -ntmpi : set the number of concurrent MPI process
  - -ntomp : set the number of concurrent OpenMP process
  - -gpu_id : set the execution on one (or several) GPU
  - -nb cpu : set the execution on only the CPU
- If no parameter is done : mdrun explores the machine
  - Process OpenMP ~ detected cores (if no OMP_NUM_THREADS)
  - Process MPI ~ detected cores (if OMP_NUM_THREADS mis à 1)
- In a multigpu launch : -gpu_id 01 activates le 0 et le 1
  - For Nvidia devices, prefer the variable : CUDA_VISIBLE_DEVICES

The Test Bench... Firstly, the 9 (GP)GPUs

- GPU for gamer Nvidia : driver version 375.82
  - GTX 1080Ti : Pascal architecture, 3584 cudacores, 1582 GHz
  - GTX 980Ti : Maxwell architecture, 2816 cudacores, 1075 MHz
  - GTX 780Ti : Kepler architecture, 2880 cudacores, 875 MHz
- GPGPU Nvidia : driver version 375.82
  - Tesla P100 : Pascal architecture, 3584 cudacores, 1126 GHz
  - Tesla K80 : Kepler architecture, 2x 2496 cudacores, 560 MHz
  - Tesla K40m : Kepler architecture, 2880 cudacores, 745 MHz
- GPU AMD : driver version 17.40 (2482.3)
  - Vega 64 : GFX900 architecture, 4096 streamprocessors, 1406 GHz
  - R9-Fury : Fiji architecture, 3584 streamprocessors, 1000 GHz
  - R9-295X2 : Hawai architecture, 2x 2816 streamprocessors, 1018 GHz

Test benchs : Their basement, the 7 CPU

- Ryzen7 1800 : 8 cores, 16HT, 3.6 GHz for R9-Fury
- Skylake i7-6700K : 4 cores, 8HT, 4 GHz
  - for Vega64 and R9-295X2
- E5-2637v4 : 2x4 cores, 16HT, 3.5 GHz for GTX1080 Ti
- E5-2640v4 : 10 virtual cores, 2.4 GHz for Tesla P100
- E5-2637v2 : 4 virtual cores, 3.5 GHz for Tesla K40m
- E5-2607v2 : 2x4 cores, 2.5 GHz for GTX980Ti
- E5- 2620 : 6 cores, 12 HT, 2 GHz for GTX780Ti

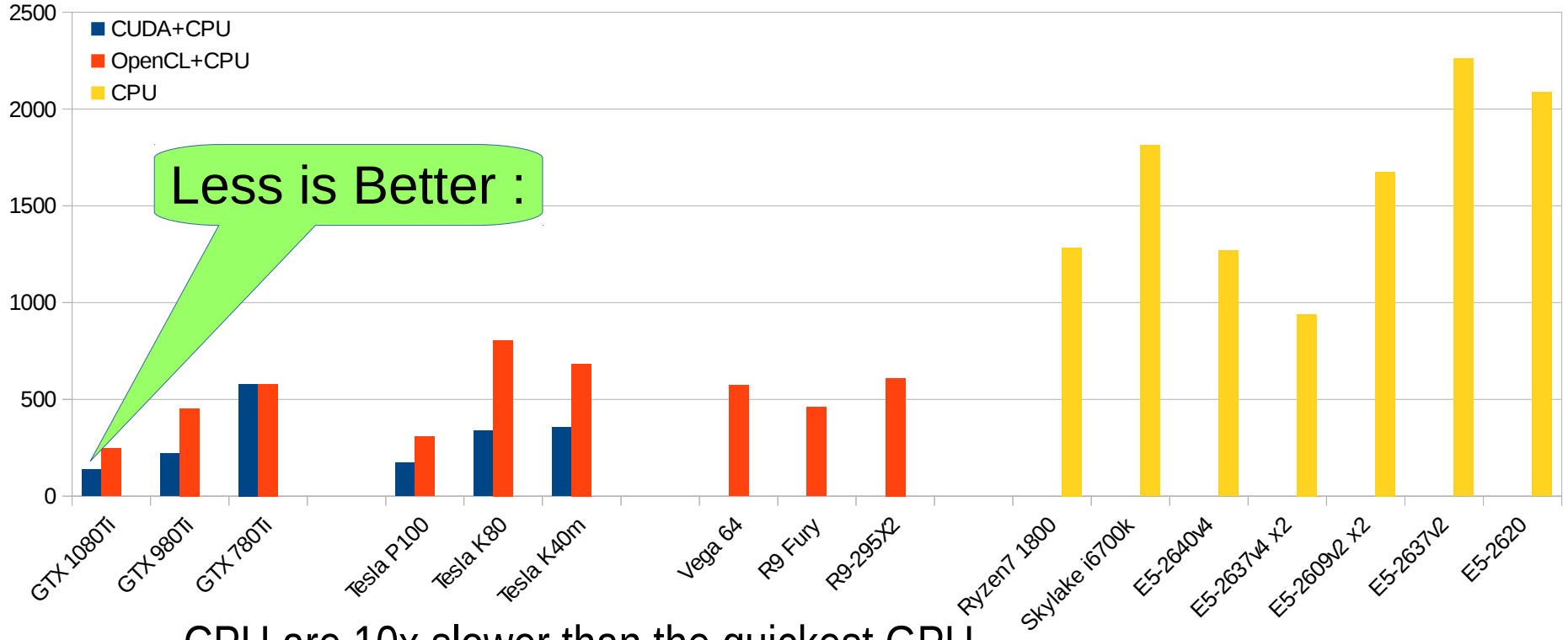
Gromacs : Use Case #1 « Test » from Nvidia

- Configuration of paths and execution folders
  - export GPU=GTX1080Ti
  - mkdir -p /scratch/Gromacs/Test/$(hostname)_${GPU}
  - cd /scratch/Gromacs/Test/$(hostname)_${GPU}
  - GRODIR=/scratch/Gromacs/2018.1_${GPU}
  - source $GRODIR/bin/GMXRC
- Loading & expansion of archive with input parameters
  - wget ftp://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz
  - [-d water-cut1.0_GMX50_bare] && rm -r water-cut1.0_GMX50_bare
  - tar -zxvf water_GMX50_bare.tar.gz ; cd water-cut1.0_GMX50_bare/1536
- Executions of launches
  - $GRODIR/bin/gmx grompp -f pme.mdp
  - /usr/bin/time $GRODIR/bin/gmx mdrun -resethway -noconfout -nsteps 4000 -v -gpu_id 0

Gromacs : use case
Nvidia Test : « time » output

- TIME Command being timed: "/scratch/Gromacs/2018.1_GTX980Ti_OpenCL/bin/gmx mdrun -resethway -noconfout -nsteps 4000 -v -gpu_id 0"
- TIME User time (seconds): 2767.16
- TIME System time (seconds): 25.98
-
- TIME Percent of CPU this job got: 616%
- TIME Average shared text size (kbytes): 0
- TIME Average unshared data size (kbytes): 0
- TIME Average stack size (kbytes): 0
- TIME Average total size (kbytes): 0
- TIME Maximum resident set size (kbytes): 4820892
- TIME Average resident set size (kbytes): 0
- TIME Major (requiring I/O) page faults: 39
- TIME Minor (reclaiming a frame) page faults: 1248968
- TIME Voluntary context switches: 288413
- TIME Involuntary context switches: 447953
- TIME Swaps: 0
- TIME File system inputs: 191072
- TIME File system outputs: 0
- TIME Socket messages sent: 0
- TIME Socket messages received: 0
- TIME Signals delivered: 0
- TIME Page size (bytes): 4096
- TIME Exit status: 0

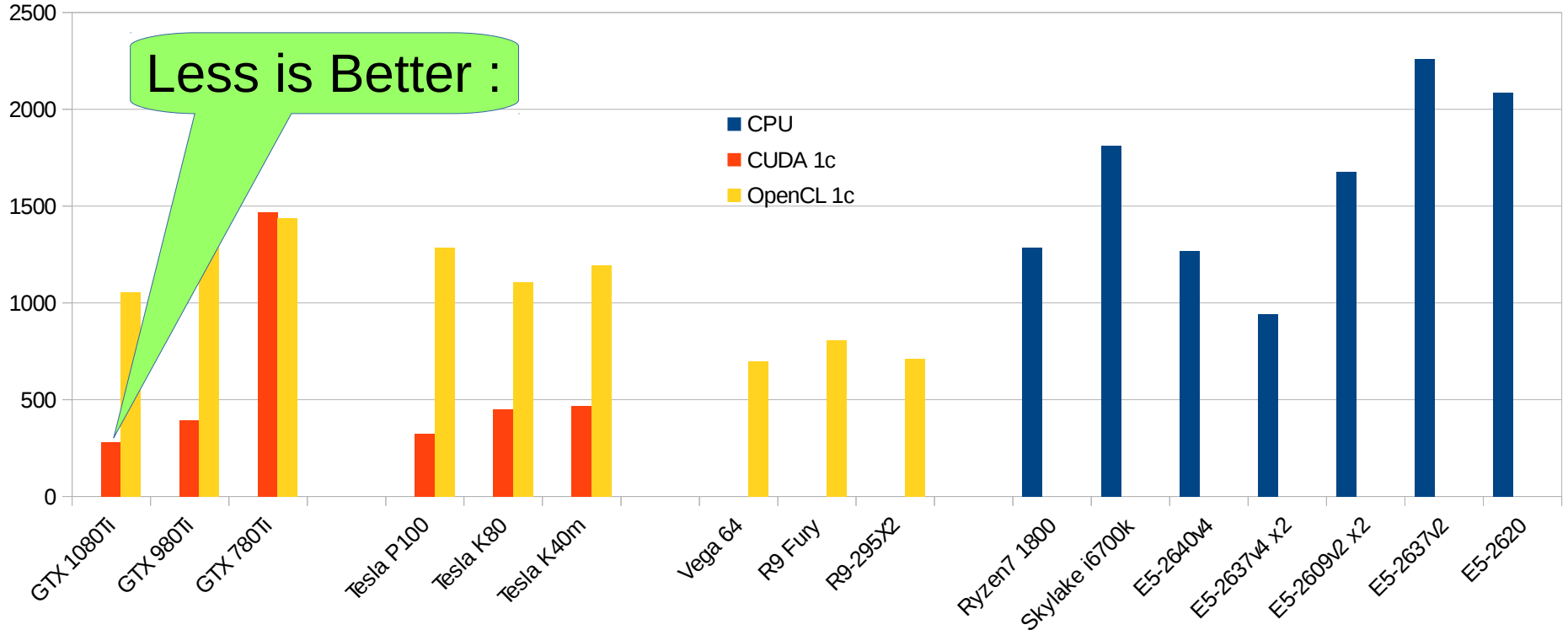
Use case #1 : Nvidia test
If we compare with CPU



- CPU are 10x slower than the quickest GPU
- GPU are at worst twice as fast as the fastest CPU

- Scam ! The code is an hybrid one ! (CPU/GPU)

Use case #1 : Nvidia test Renormalisation : GPU on 1 core...



  - The fastest GPU with 1 core is 4x fastest than the fastest CPU
  - GPU are in all cases faster than any assembly of CPU
- So, to use GPU for Gromacs, it's useful...

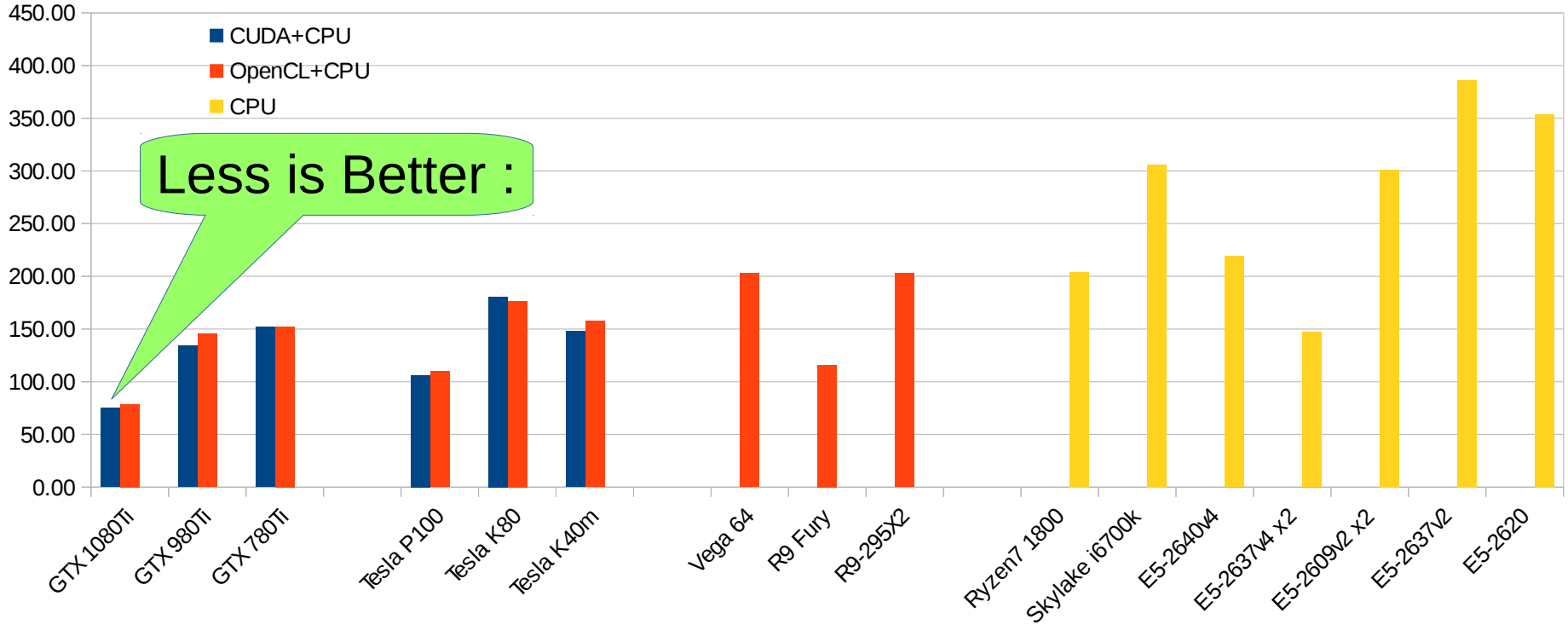
Gromacs : Test case #2
« Test » Tutorial on Free Energy

- Configuration of paths and execution folders
  - export GPU=GTX780Ti
  - export GRODIR=/scratch/Gromacs/2018.1_${GPU}
  - source $GRODIR/bin/GMXRC
  - cd /scratch/Gromacs/Test ; mkdir $(hostname)_${GPU} ; cd $(hostname)_${GPU}
- Loading & expansion of archive for input parameters
  - tar xzf ../gromacs-free-energy-tutorial.tgz ; cd gromacs-free-energy-tutorial
  - SIZE=10
- Execution of tasks
  - $GRODIR/bin/gmx editconf -f ethanol.gro -o box.gro -bt dodecahedron -d $SIZE
  - $GRODIR/bin/gmx solvate -cp box.gro -cs -o solvated.gro -p topol.top
  - $GRODIR/bin/gmx grompp -f em.mdp -c solvated.gro -o em.tpr
  - GMXLOG=GMXFE-$(hostname)-$SIZE-$(date "+%Y%m%d%H%M").log
  - /usr/bin/time $GRODIR/bin/gmx mdrun -gpu_id 0 -v -deffnm em >$GMXLOG 2>&1
  - egrep time $GMXLOG | grep -v timed

Gromacs : test case #2
« Test » Tutorial Free Energy

- A more compact output :
  - TIME User time (seconds): 909.79
  - TIME System time (seconds): 4.95
  -

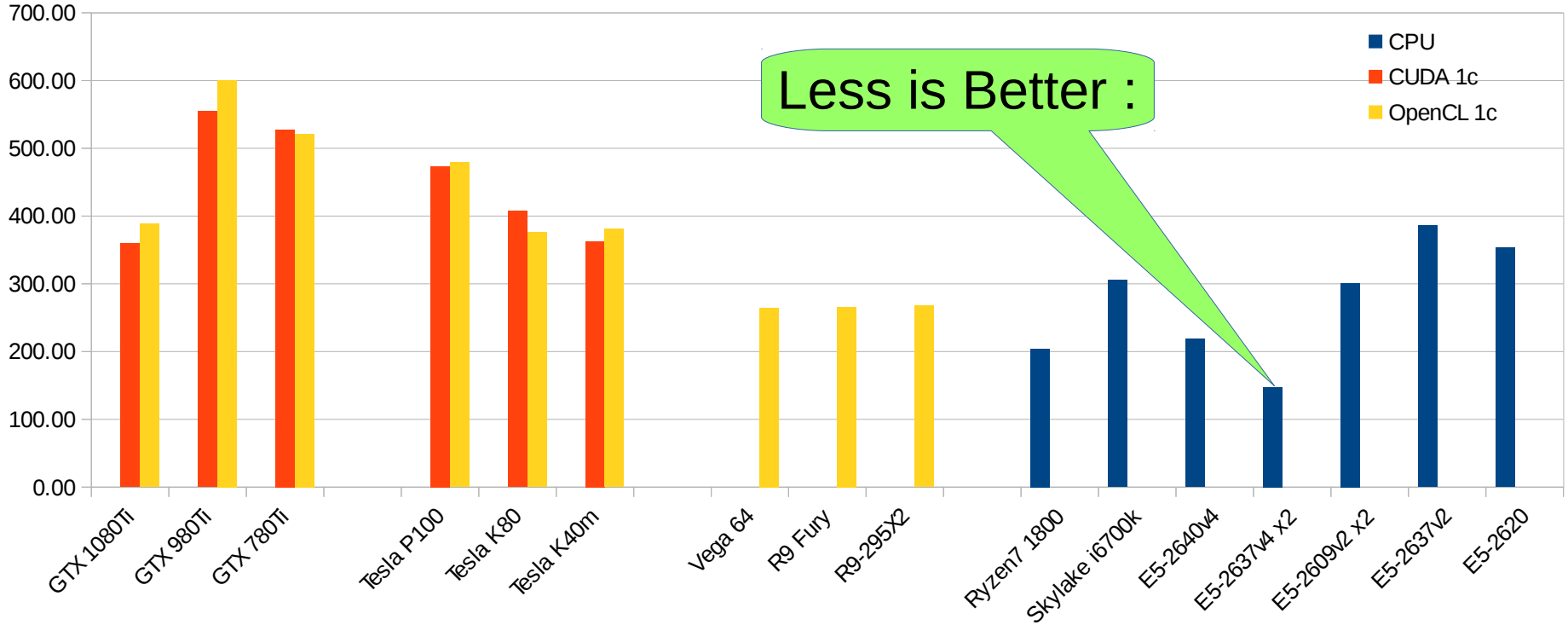
Use case #2 : test Nvidia
After, we add the CPU



- CPU are only 6x slower than the fastest GPU
- Slowest GPU are slower than the fastest CPU !

- And always the scam ! The code is an hybrid one (CPU/GPU) !

Use case #2 : Nvidia test Renormalisation : GPU on 1 core...



- AMD GPU are fastest than all Nvidia GPU
- CPU are generally fastest than GPU

- So, using GPU for Gromacs & this use case, be careful !

Gromacs : conclusion

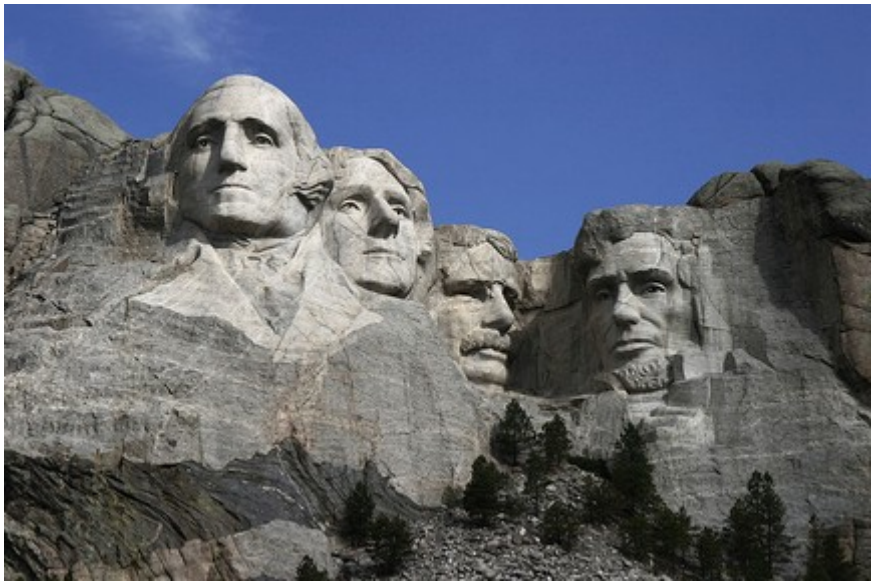
- GPU powerful, BUT in the RIGHT context
  - Number of cores & speed of memory are important
- CUDA always very efficient but with huge CPU
- OpenCL in progress
  - But 2x slower than the CUDA one in som cases
- Using Gamer GPU cartes de Gamer relevant
  - But for computing in single precision
- Basically, without experimentation, no optimal use…

But where is the 100 speed'up of the advertisement ?

For «business» applications « Ikea » or « Crozatier » models

- Be careful !
  - Today is not tomorrow : retreive all context of execution
    - Hardware : lshw, lscpu, nvidia-smi, … & software : OS, drivers, etc...
  - For each use (ou preparation) its optimal context
    - A cooking, it's the assembly between recipie, utensils et ingredients
- Don't worry !
  - It has always been the case CPU ;-)
- Let's return to the other codes :
  - « integrator » approach or « developper » approach

Develop or integrate ?



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).

Between integrator and developer Que choisir ?

- 2 approaches
  - « Integrator » approach
    - The code uses generic librairies
    - The code is modified by changing the « calls » to external libraries by others
  - « Developer » approach
    - The GPU is a new device
    - It needs a learning as every new material
    - The code must be rewritten to use as well as it's possible
- 1 constraint but 2 ways : to reduce time
  - Human Time of programming : almost integrator
  - Elasped time of execution : almost developper

Common uses...

- For Linear algebra from « basic » to « advanced »
  - Basic with BLAS
  - Advanced with LAPACK
- For Linear algebra on sparse systems
  - UMFPack
- For spectral analysis with Fourier Transform
  - FFTPack
- For differential equation solving
  - PetSC
  - Trilinos

Software evolutions matching technological evolutions

- In dense linear algebra :
  - BLAS with OpenBLAS, MKL (OpenMP, MPI), BLACS (MPI)
  - LAPACK with ACML (OpenMP), Scalapack (MPI)
- In sparse linear algebra :
  - MUMPS (OpenMP, MPI) but depends of BLAS
- In spectral analysis :
  - FFT with FFTw3 (OpenMP), FFTw2 (MPI)

Consistent to see specific GPU versions !

For linear algebra : profusion

- Fonctions BLAS :
  - CuBLAS : complete (but different efficiency)
  - ClBlas : not complete
- Fonctions « sparse »
  - CuSparse
- Fonctions LAPACK :
  - Cula : adaptation LAPACK avec CuBLAS (but old CUDA 6)
  - Magma : active (version 2.5.0 de 2019Q1)
- API Python : everything for the « glue » language...

For the Fourier Transform

- CuFFT : for C, C++, Fortran
  - Float on 32, Double on 64, Complex on 32 and on 64
  - 1D, 2D, 3D
  - Mixte precision : S2C, D2Z,
- clFFT : for C, C++
  - Float on 32, Double on 64, Complex on 32 and on 64
  - 1D, 2D, 3D
- Reikna for Python

How to program in // ?
The different librairies

Parallel programming models : overview

	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*

Librairies de programmation parallèle

	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 :

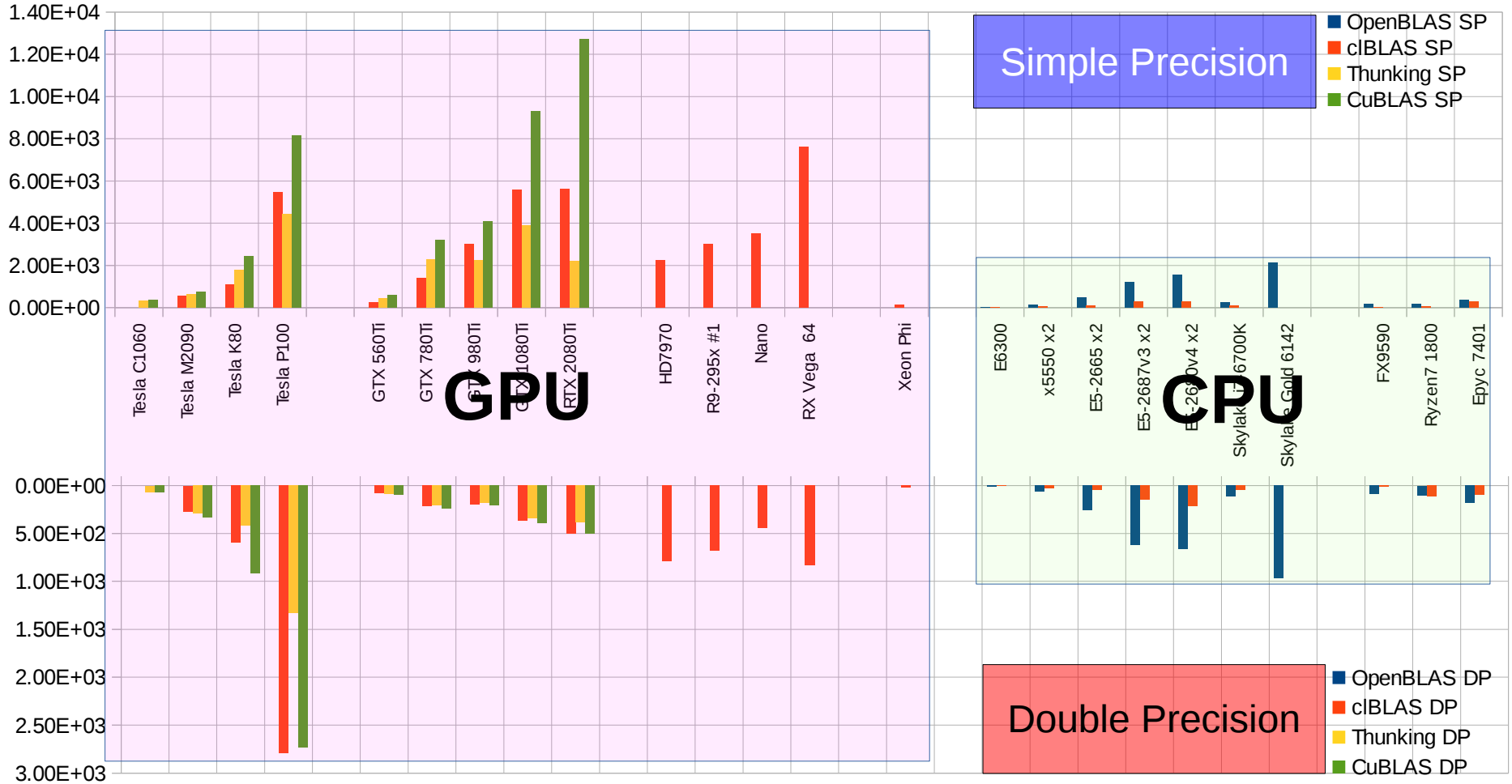
$$C_{ij} = \sum_{k=1}^{k=P} A_{ik} B_{kj}$$

- The C_{ij} are independant : « coarse grain » parallelism
- The $A_{ik}B_{kj}$ groupables 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 pour le produit de matrice
- The functions of all our attentions :
  - sGEMM & dGEMM for the product simple & double precision

Matrix Matrix Product Really So powerful ?

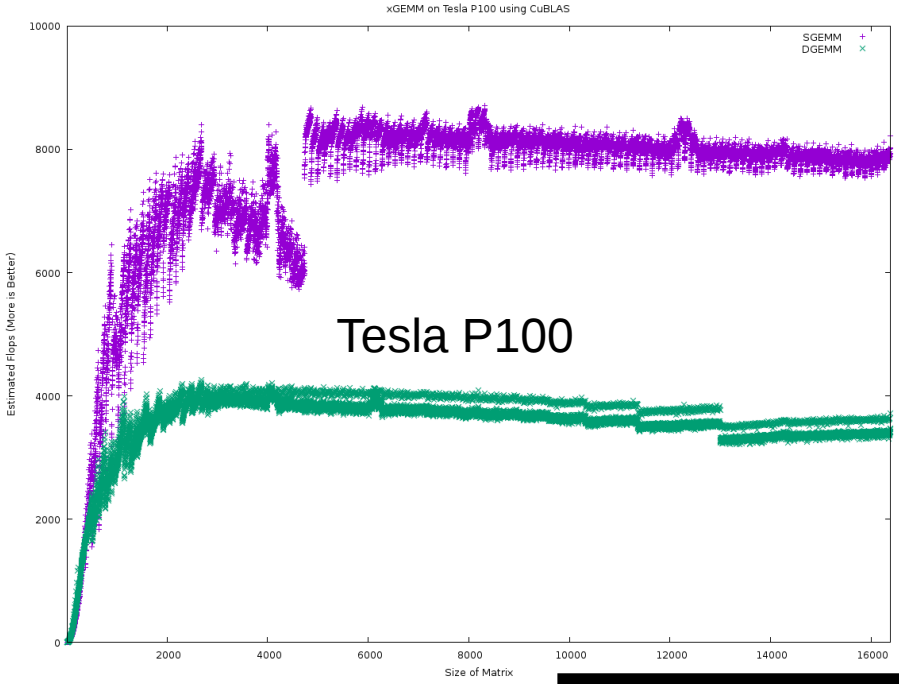


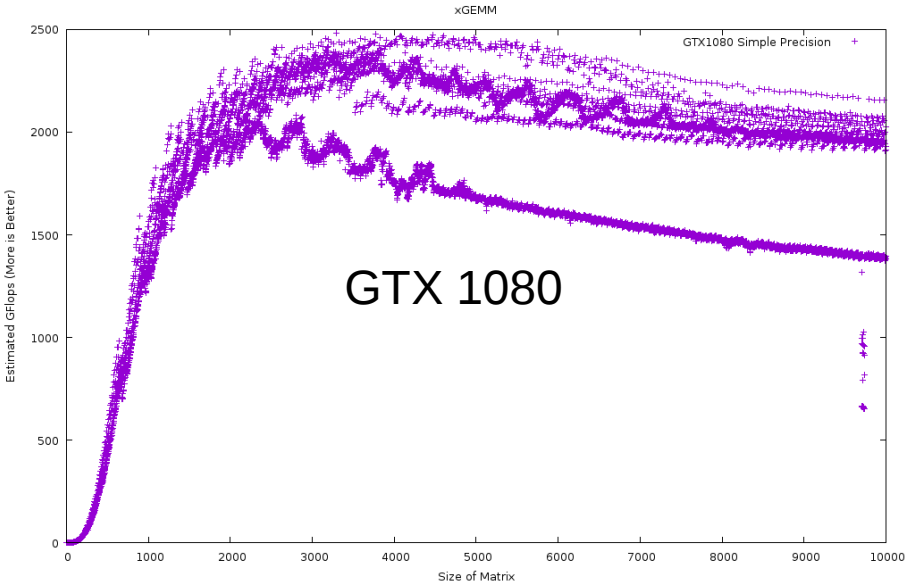
Already in 2010, strange… The behaviour of Tesla C1060...

- Solely the matrix product : xGEMM
- Property : Transpose (A * B)=Transpose(A) * Transpose(B)
- Results (in Gflops) : Yesss !
  - SP : FBLAS/CBLAS : 12, CuBLAS : 350/327 : x27 !!!
  - DP : FBLAS/CBLAS : 6, CuBLAS : 73/70 : x11 !
- Surprise : Ouch !!! CuBLAS prefers the multiples x16 !
  - SP : 16000^2, 350, but 15999^2 or 16001^2, 97 : /3,6 !
  - DP : 10000^2, 73, but 9999^2 or 10001^2, 31 : /2,35

What Nvidia nevers precises… xGEMM for different sizes...

Performance

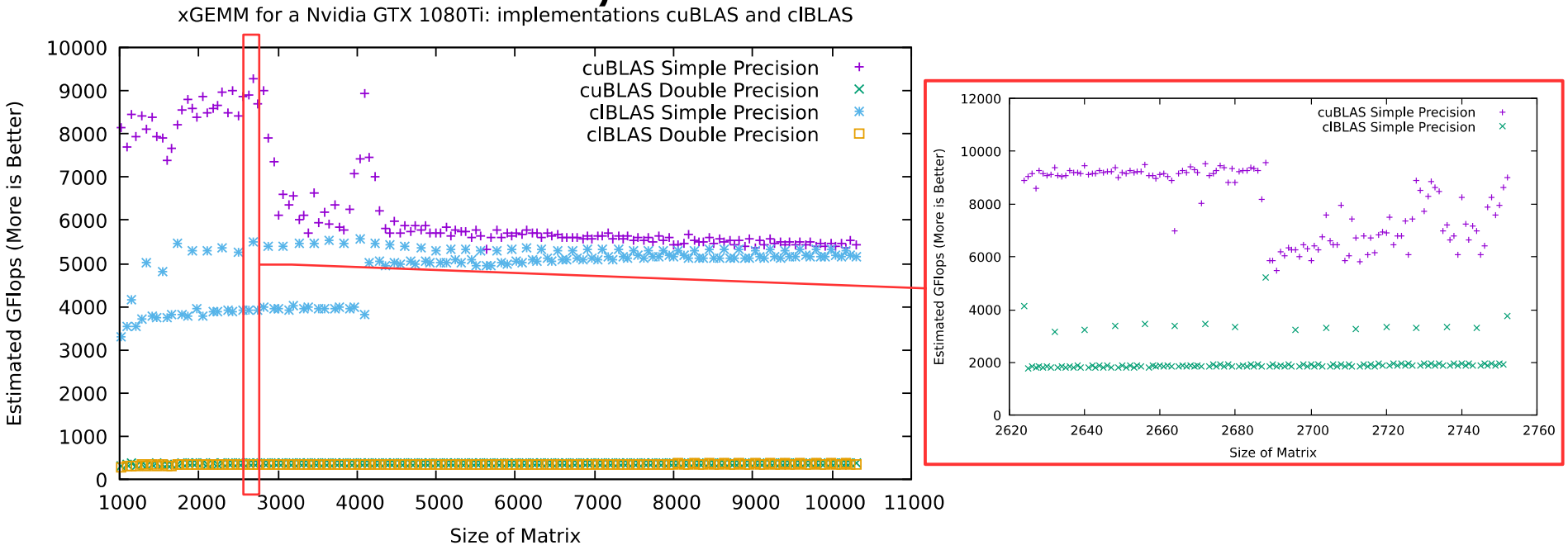




Size

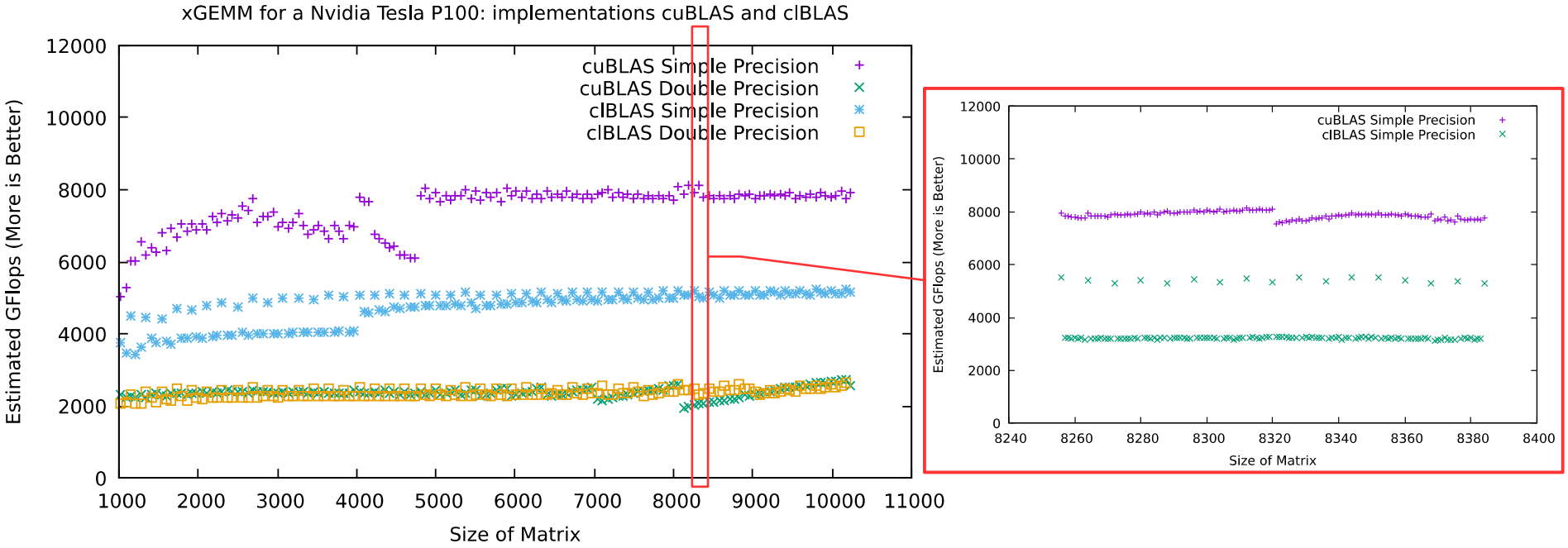
Yes, there is the raw power, but for specific sizes...

And for a « Gamer » GPU Nvidia GTX 1080Ti, a max around 2688



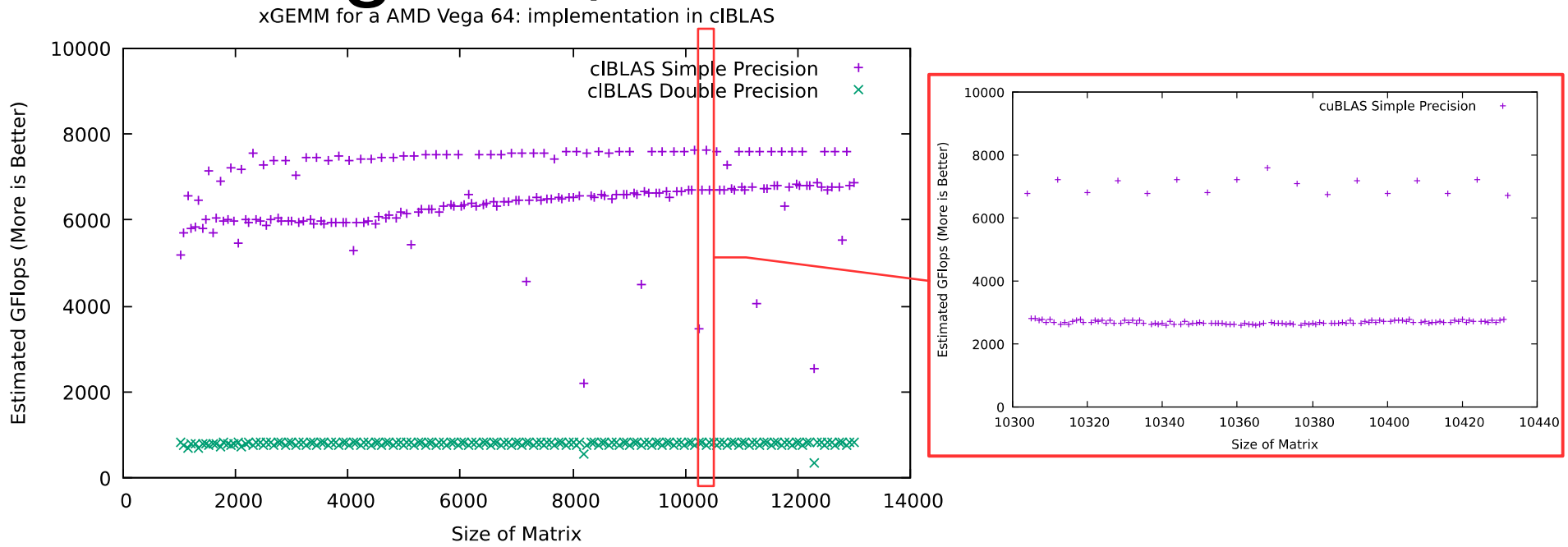
- A factor from 20 to 30 between simple & double precision
- A clBLAS implementation around the half for small sizes, equivalent after
- Threshold effects (>2688 for example)
- Arithmetic effects (prime numbers, for example 2671 or 2713)

For the « very expensive » Nvidia Tesla P100, a max for 8320



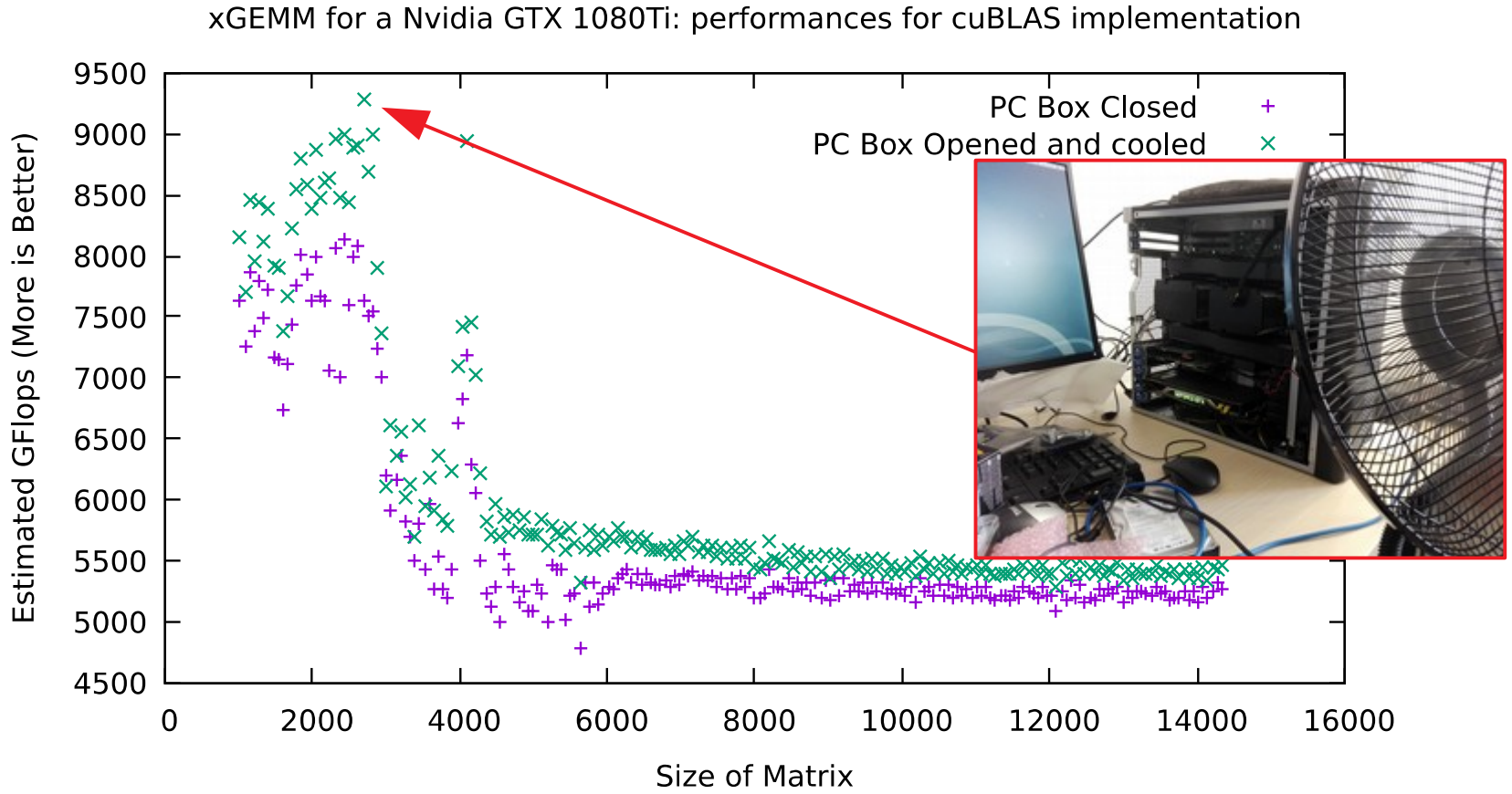
- A factor around 4 between simple & double precision
- A clBLAS implementation clBLAS with strange periods
- Several threshold effects (>4096 for exemple)
- Less arithmetic effects

And for our « huge » AMD Vega 64, a max for 10368



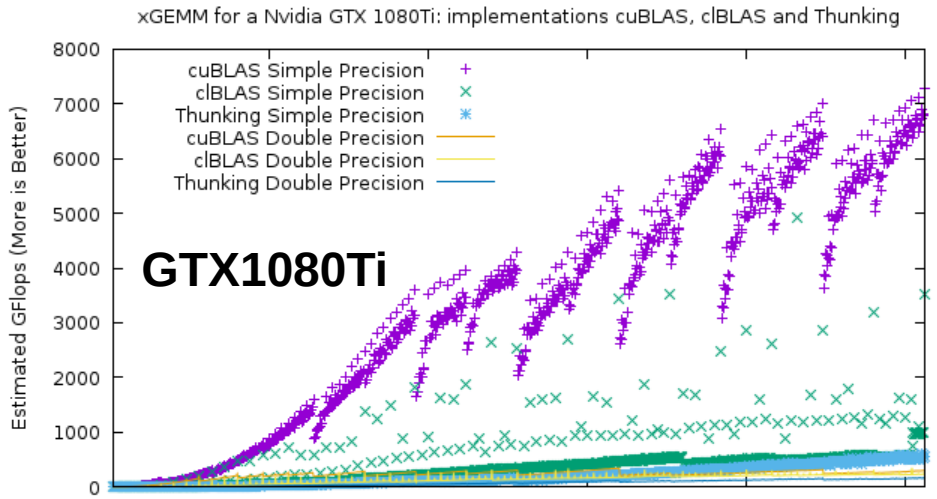
- A factor between 4 and 10 (median at 8) with simple & double precision
- Bimodal periodicity
- Less arithmetic effects

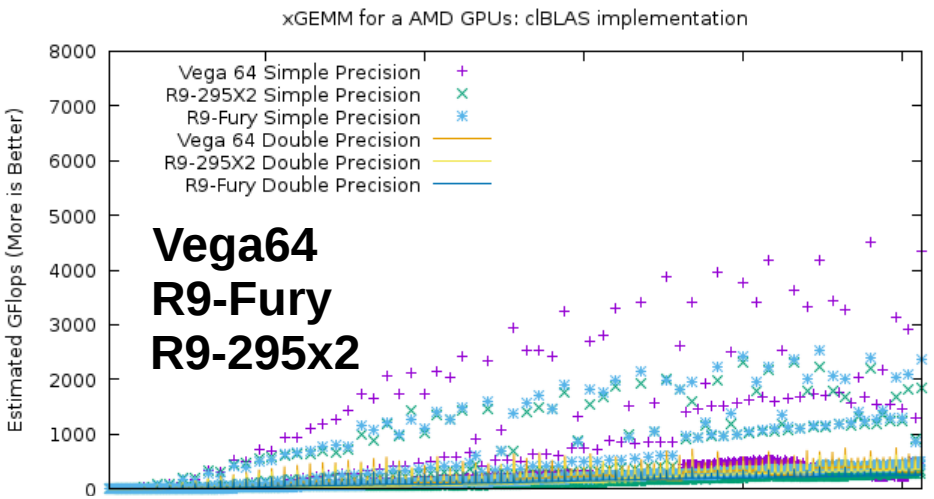
But there is worse ! During the tests of the GTX 1080 Ti

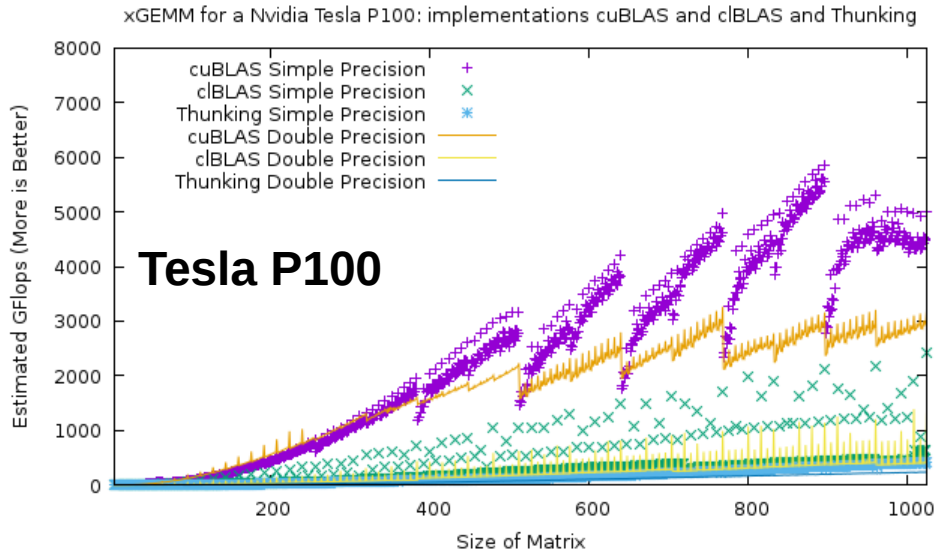


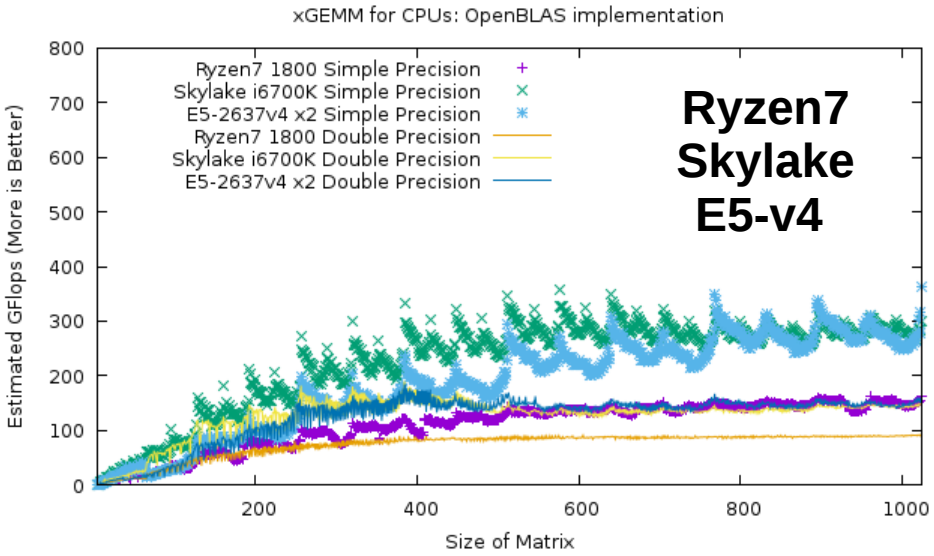
- Same workstations, boards, systems, and 20 % of difference !
  - Climatic conditions are important during experimentation...

xGEMM for the « small sizes »









Yes ! CuBLAS or clBLAS, it's nice, already in DP, but...

Preliminary conclusion on BLAS Using xGEMM

- GPU much powerful compared to CPU
  - But be careful to the no continuity of performances
- The newer is the family of GPU, the better
- ClBLAS implementation credible
  - But only for several sizes...
- Question : what about other BLAS functions ?
  - Yes, it’s better on GPU but only on big sizes (>1000)

Going down a new step « Developer » approach

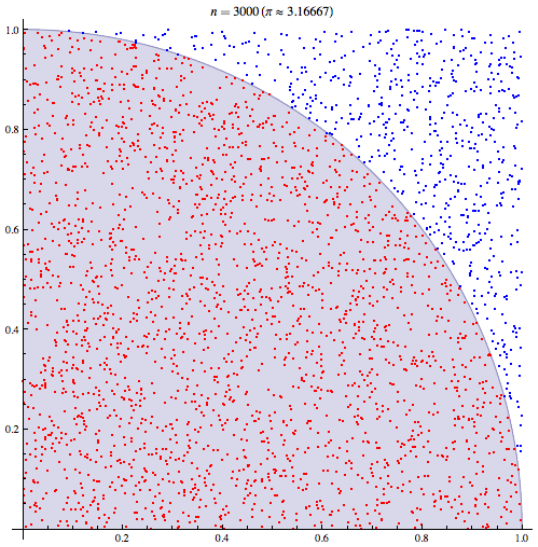
- Until now (for our presentation) :
  - « Business » code : speed'up from 2 to 5
  - « integrator » approach : with optimized librairies, factor 10x
- Now, which codes to « touch the beast » :
  - A « coarse grain » code, code « ALU » : **Pi Dart Dash**
  - A « fine grain », code « memory » : **Nbody**
  - A code « memory grain » : **Splutter**
- 2 goals : to compare GPU vs CPU & GPU vs GPU

OpenCL environment : C but 11 implementations on x86

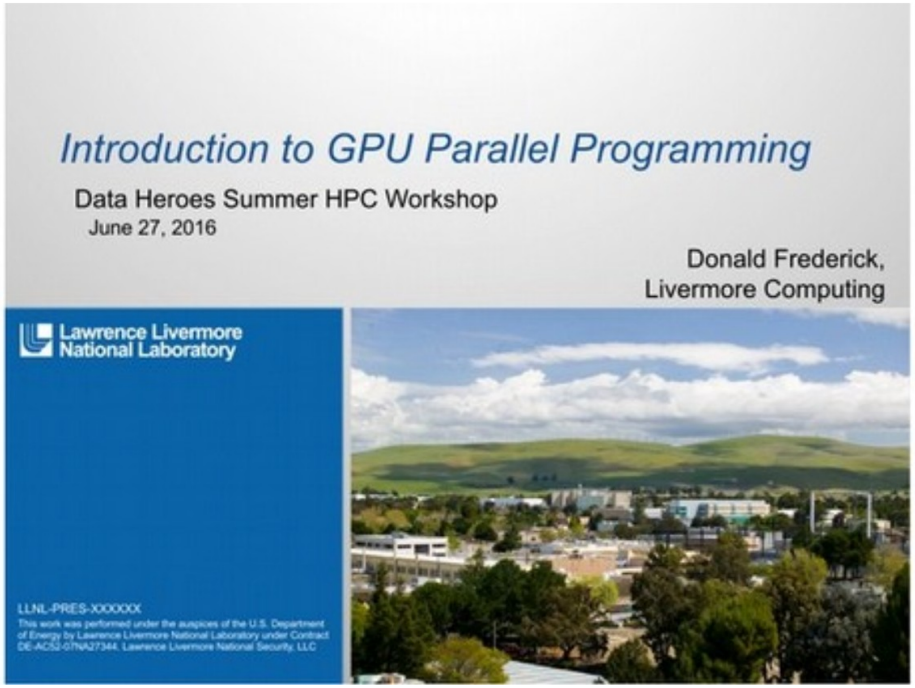
- 3 implementations for CPU:
  - AMD one : the historical one ...
  - Intel one : very efficient (but not always, for recent processors and AMD one)
  - OpenSource one : PortableCL (POCL)
- 4 implementations for GPU:
  - AMDgpu-pro : for their recent GPU, very tricky to use...
  - Nvidia: only for the eponymous GPU
  - Beignet : Open Source version for the Intel GPU (integrated besides CPU on socket)
  - 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
- 1 implementation for FPGA :
  - Altera, now provided by Intel

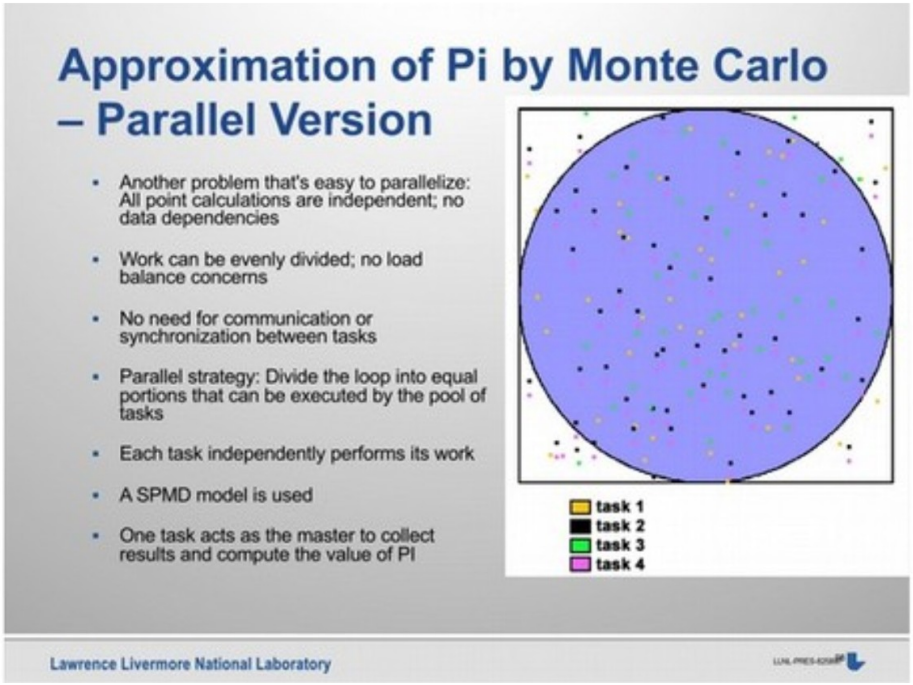
And for a simple implementation ?
PiMC : Pi by Dart Board Method

- 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

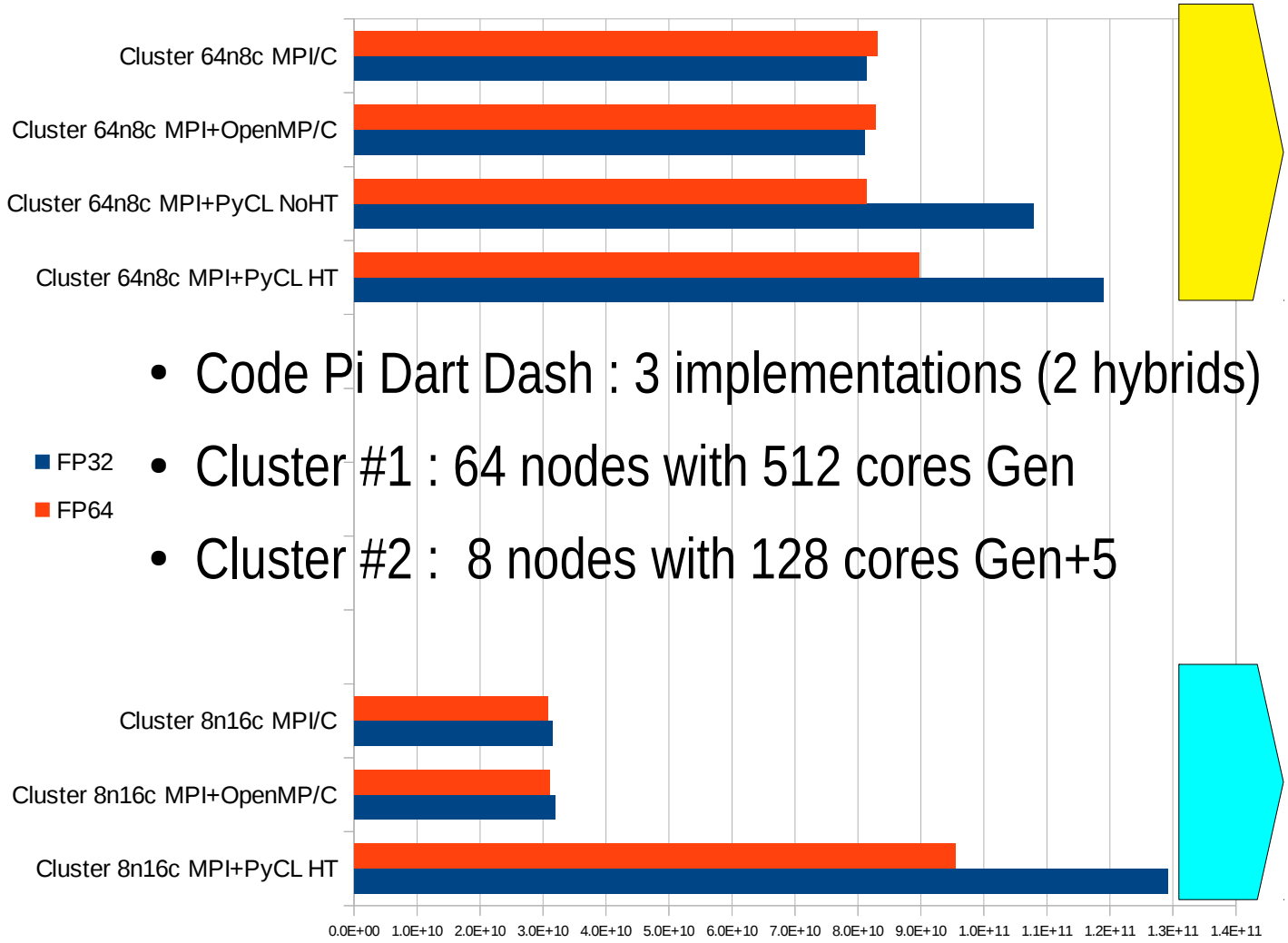


Not very relevant as code
Just have a look to a LLNL tutorial...

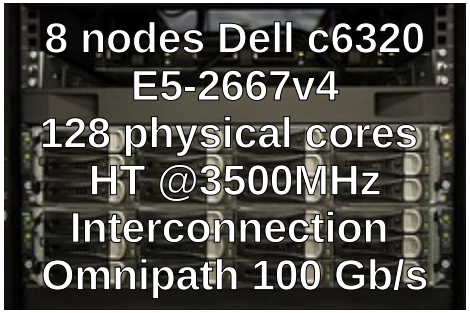




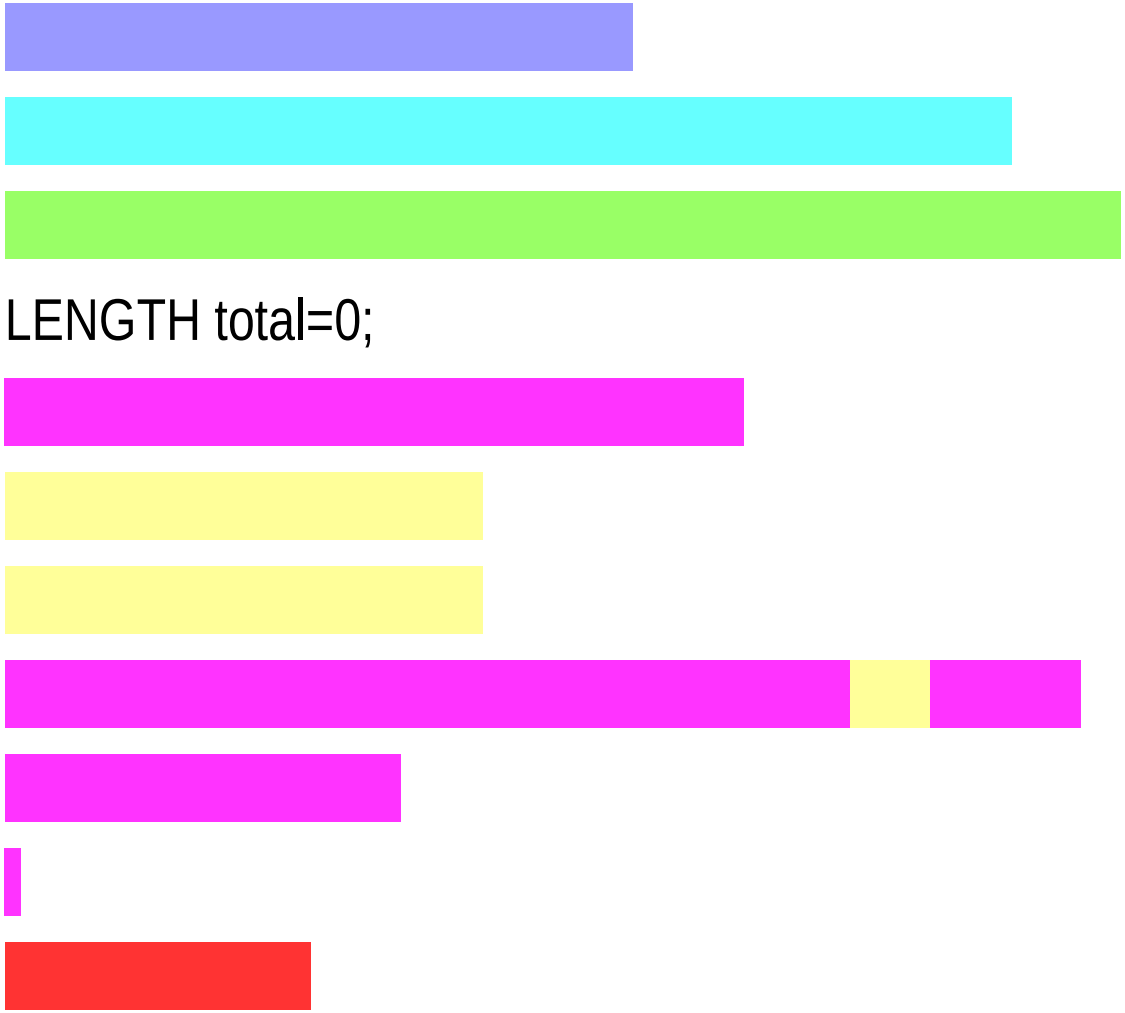
On clusters ? Python efficient ? A small comparison with GNU



- 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

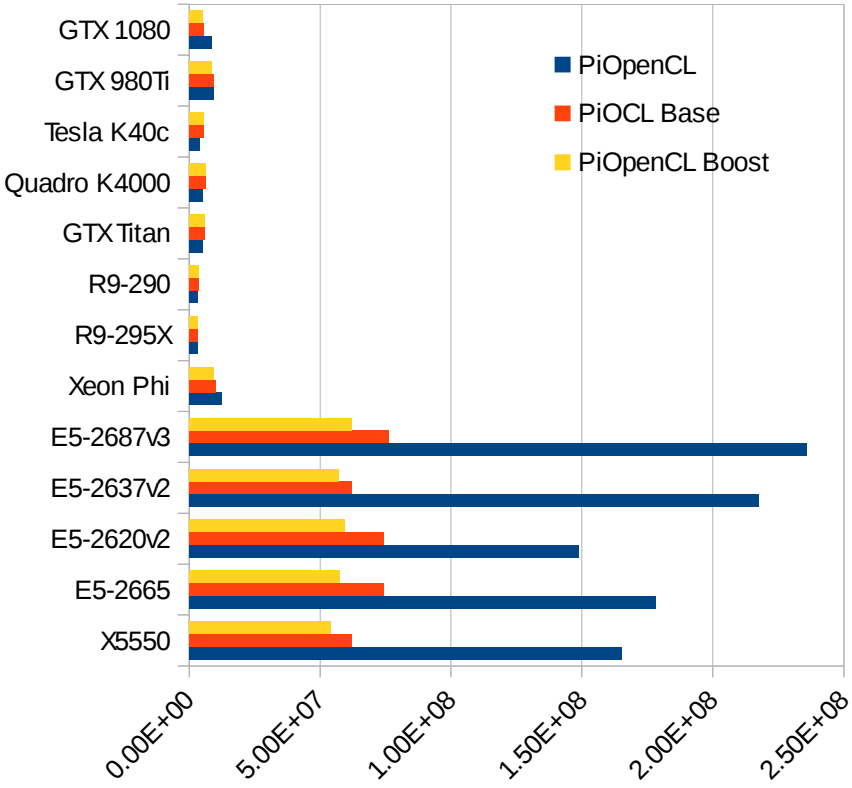


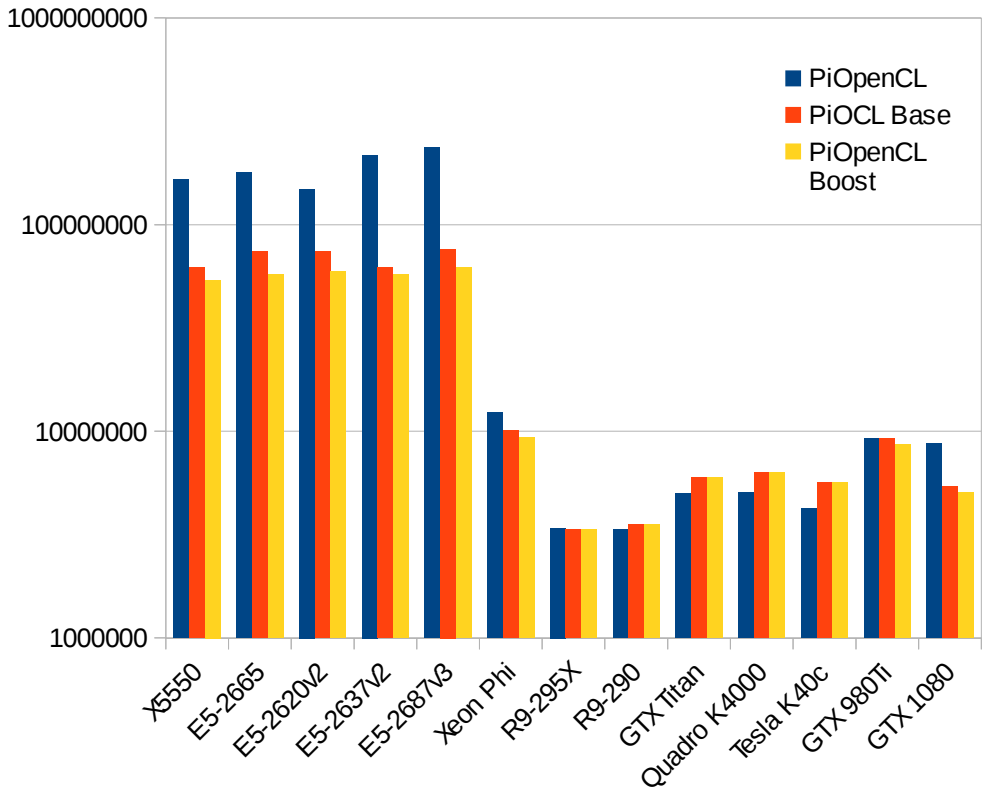
A very monolithic version… Variable & RNG fixed…



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

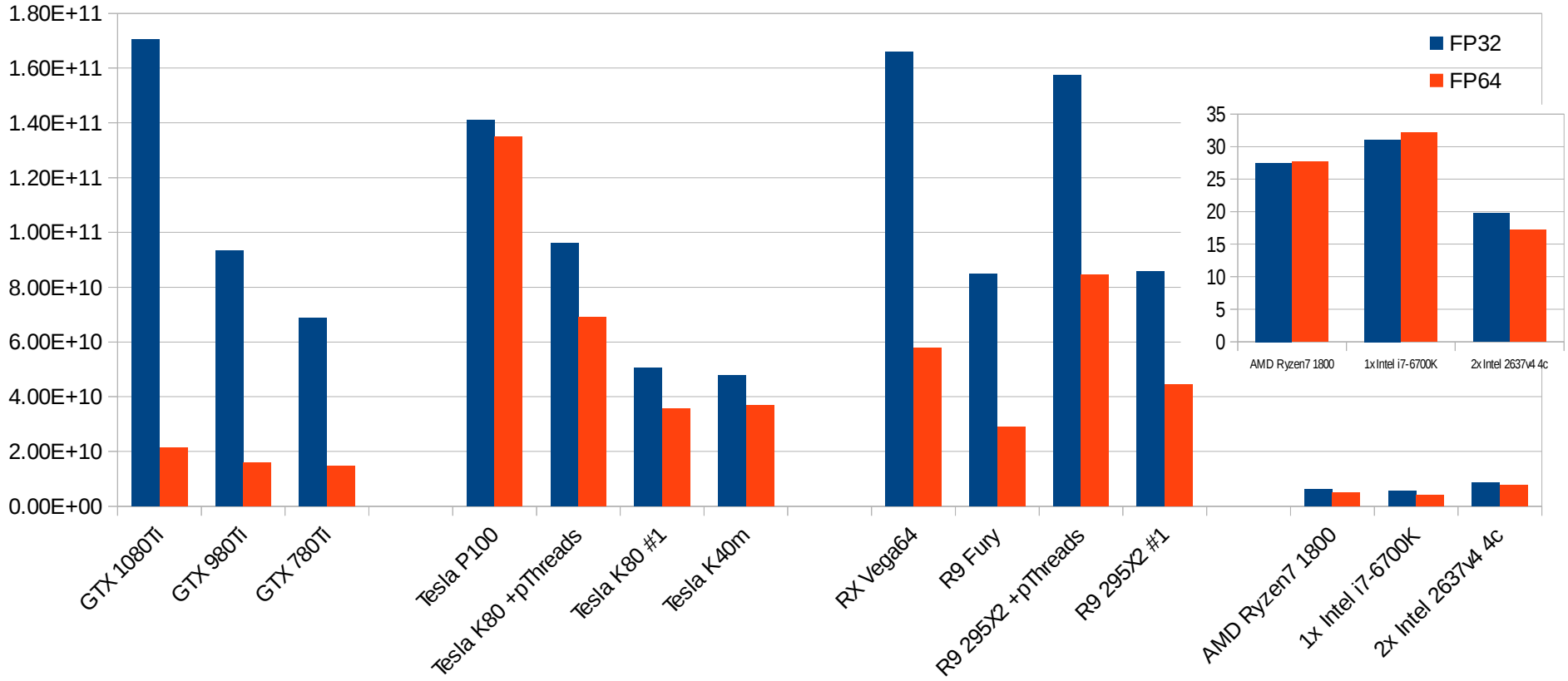
From 20 to 50x slower !





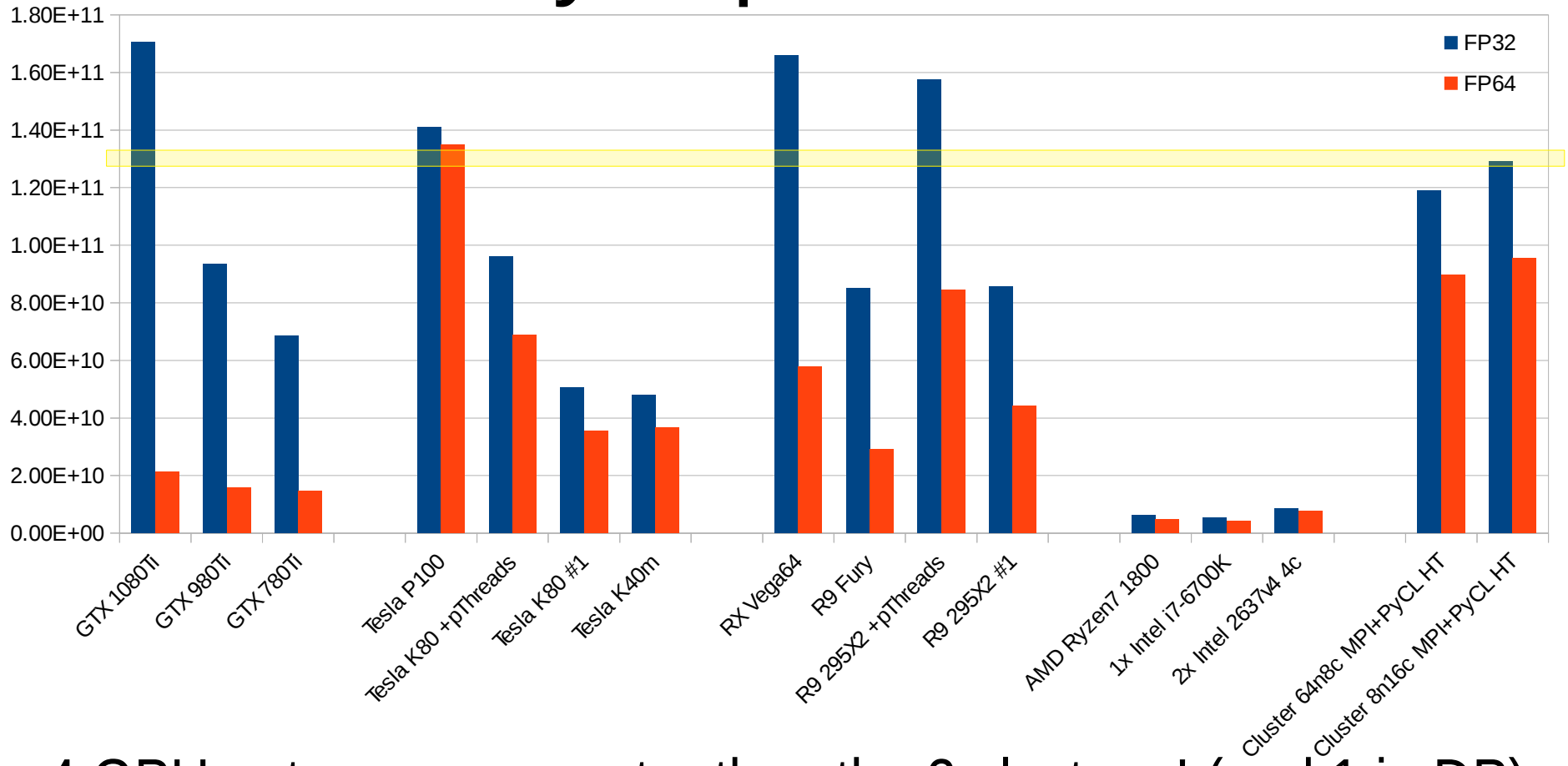
1.5 order of magnitude

Pi Dart Dash, for our material… With a multiGPU Python OpenCL



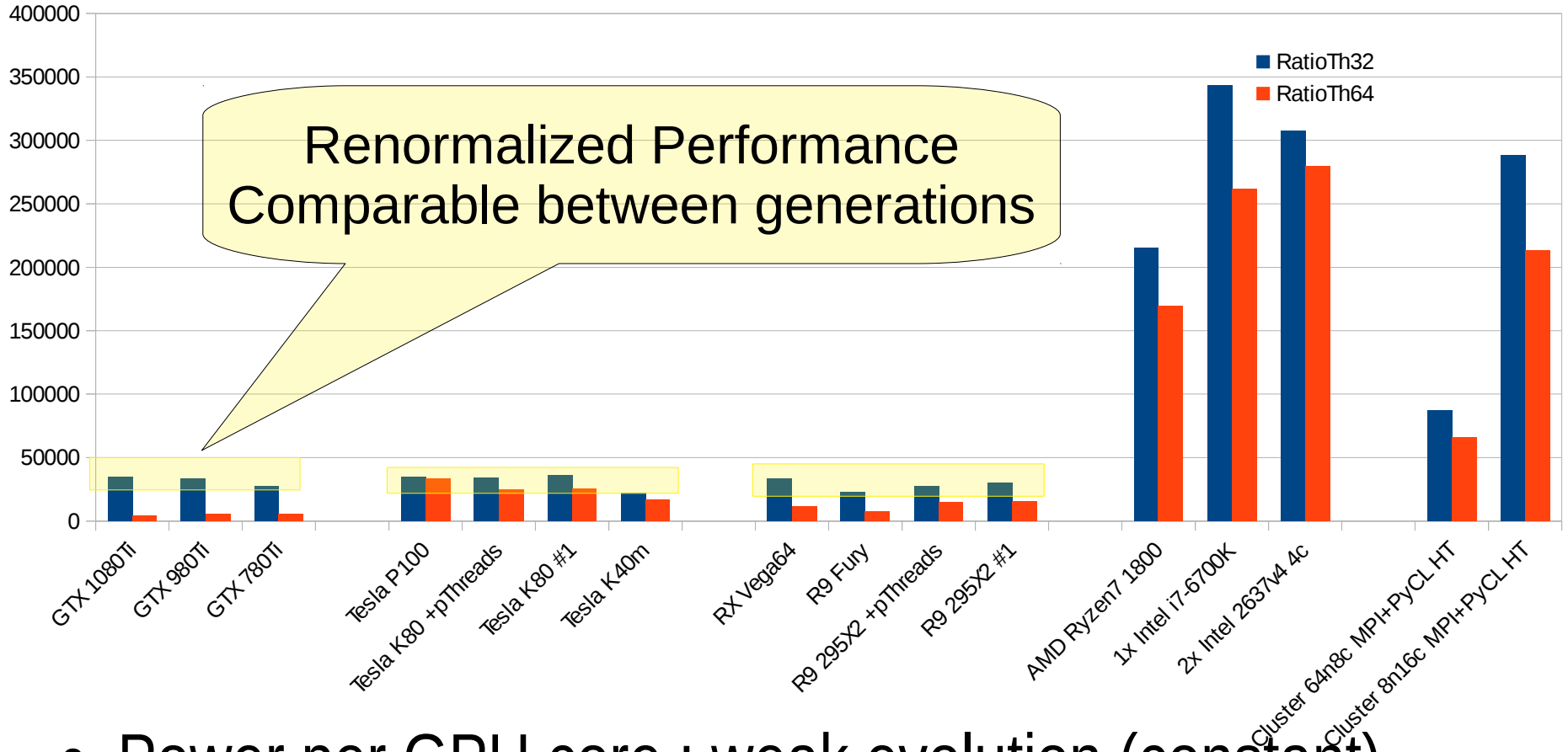
And integrating our clusters ?

Pi Dart Dash, clusters added… Very impressive !



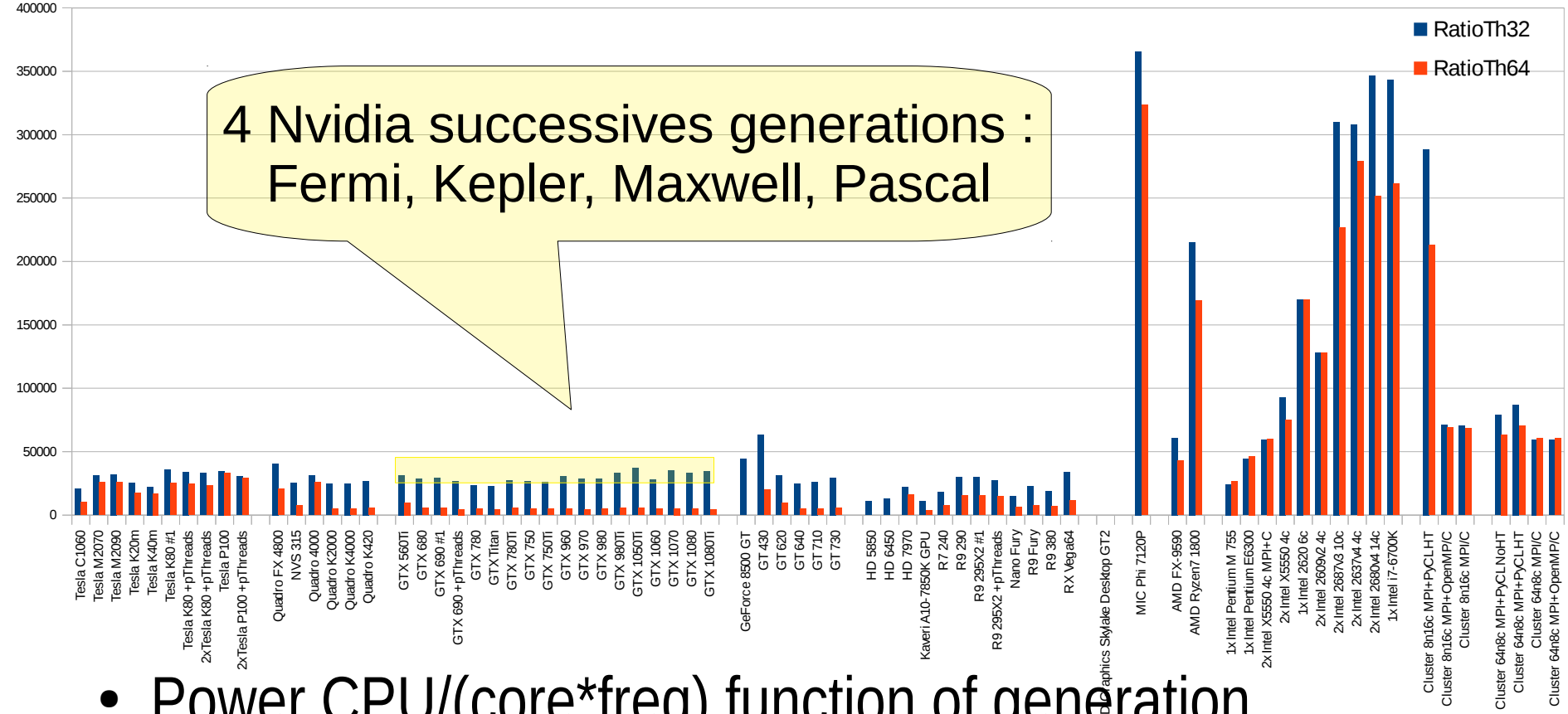
4 GPU got a power greater than the 2 clusters ! (and 1 in DP)

If we renormalize to « theoritical » power ? Itops on cores*frequency



- Power per GPU core : weak evolution (constant)
- Power per CPU core inscreases !

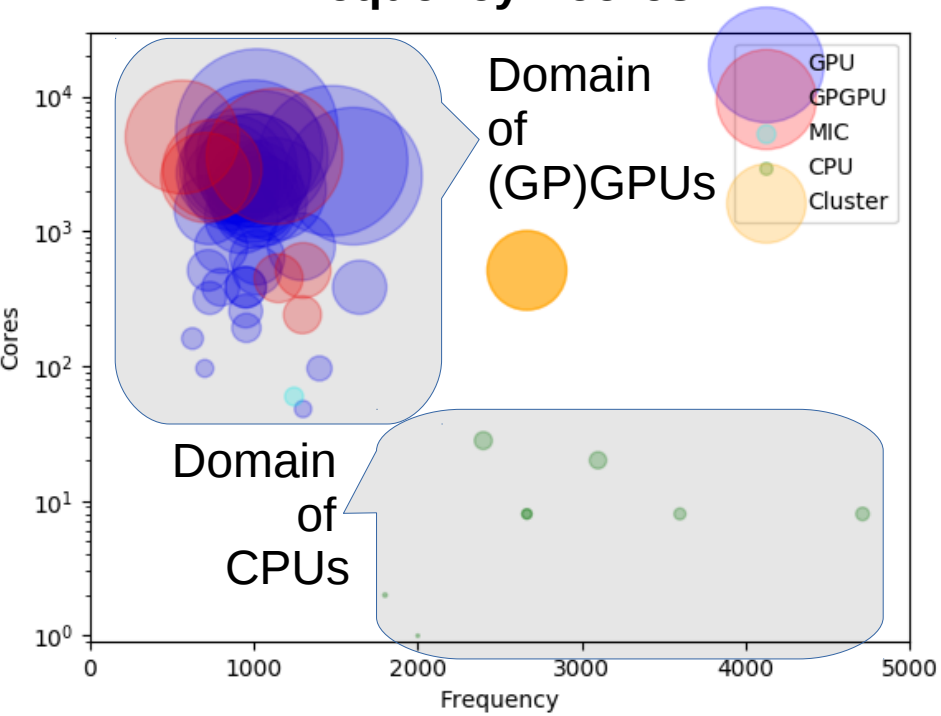
Renormalized power CPU/GPU Can it considered as a law ?



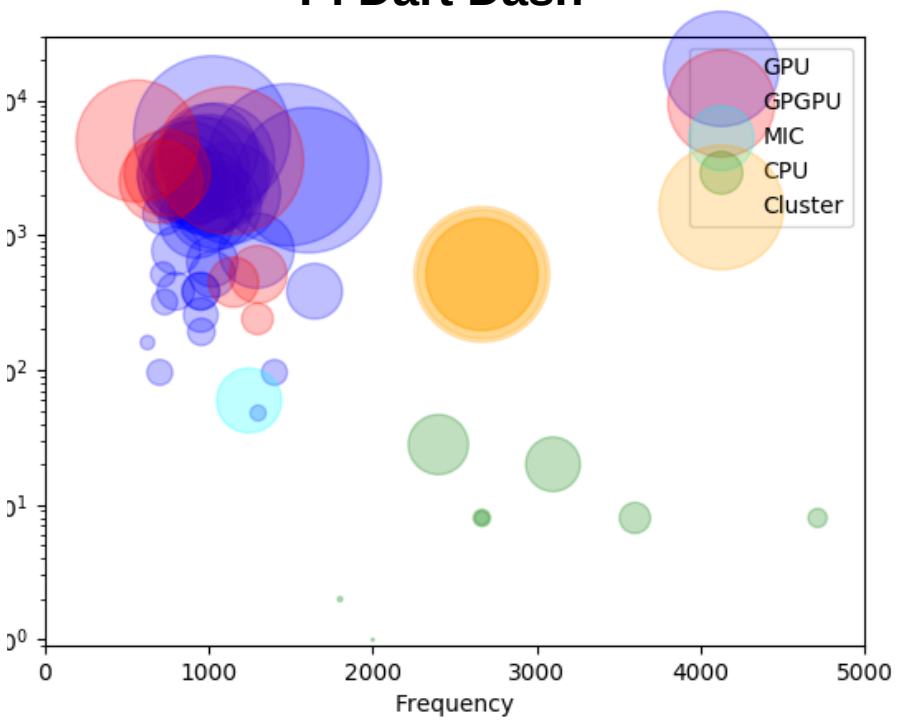
- Power CPU/(core*freq) function of generation
- Power GPU/(core*freq) independant

Represent the performances... Question of beats, cores ?

Theoritical Performance
Frequency * cores

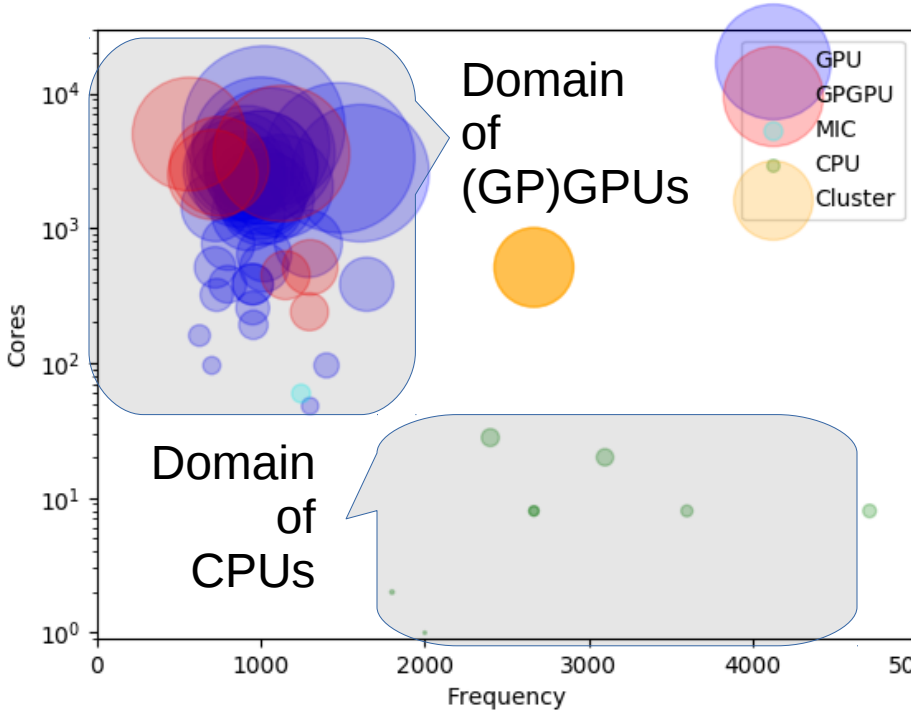


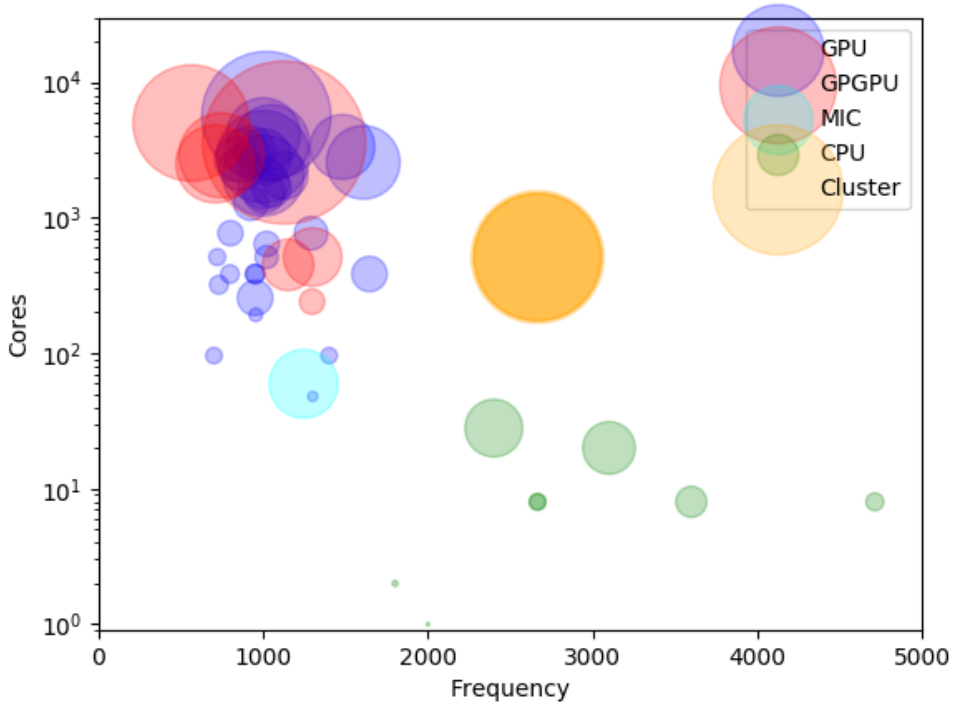
Performance in 32 bits
“Pi Dart Dash”



- On a GPU, consistency between theoritical & practical performances
- On a CPU, relative performance better (more IPC on CPU than GPU)

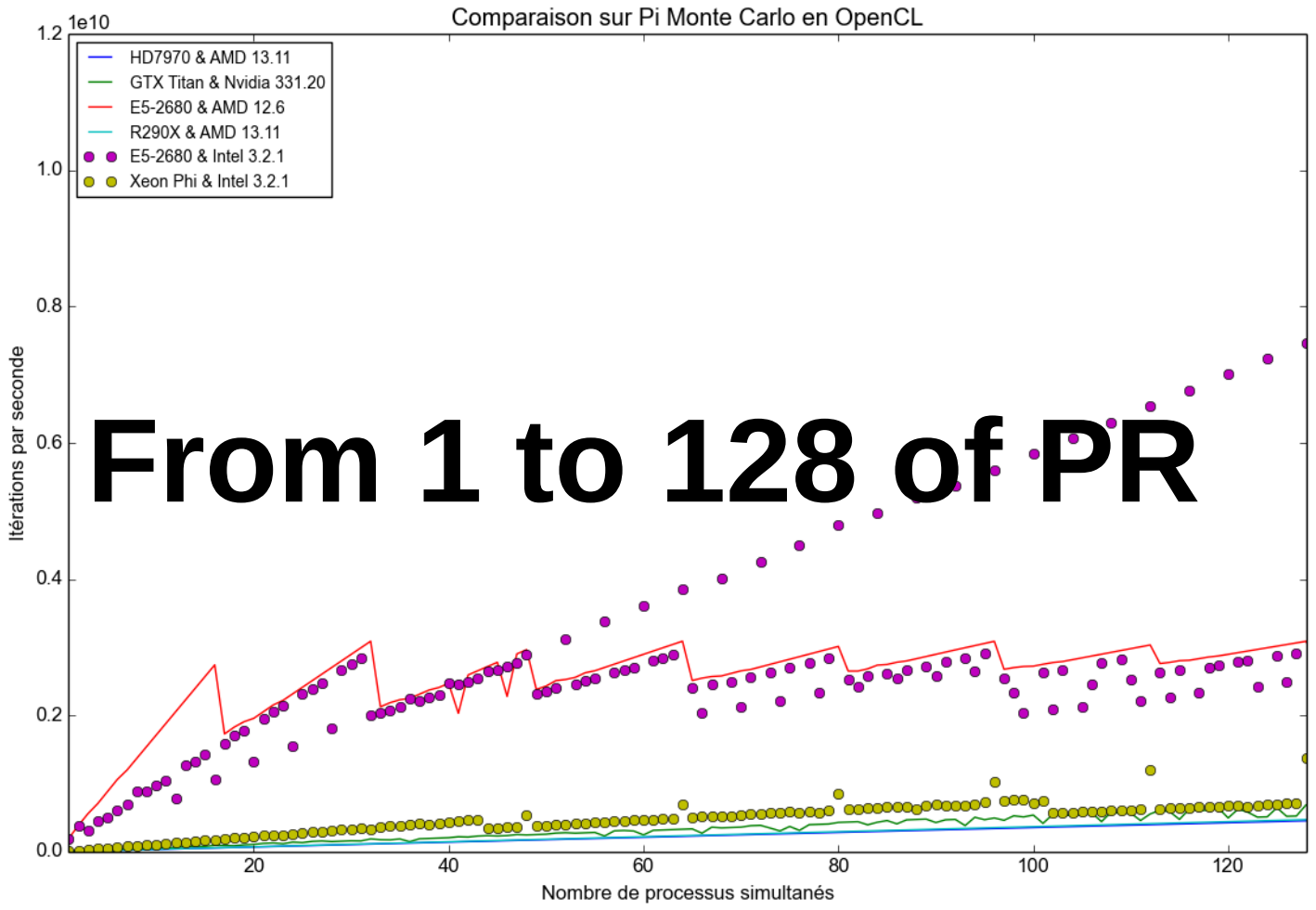
So performance in IT systems
Question of beats, cores, and bits!



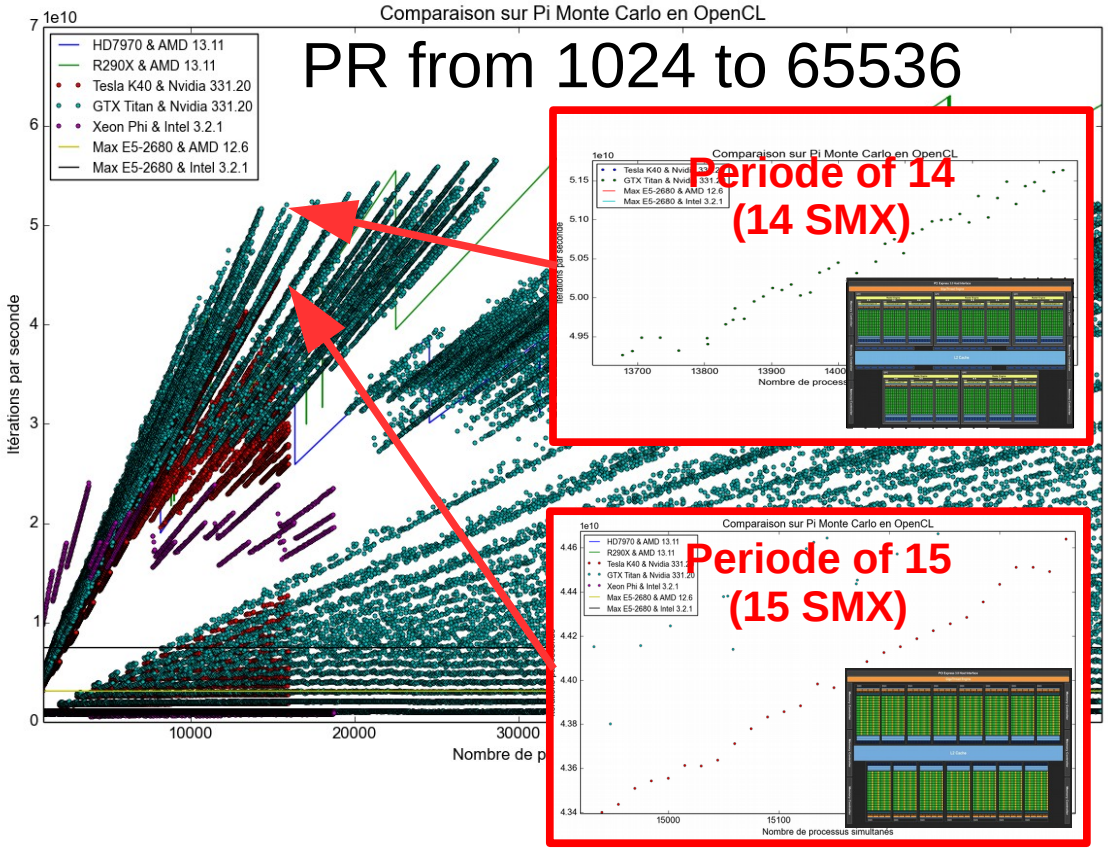


- On GPUs, performances lower dramatically for gamer GPU in 64 bits
- On CPU, performance relative encore meilleure

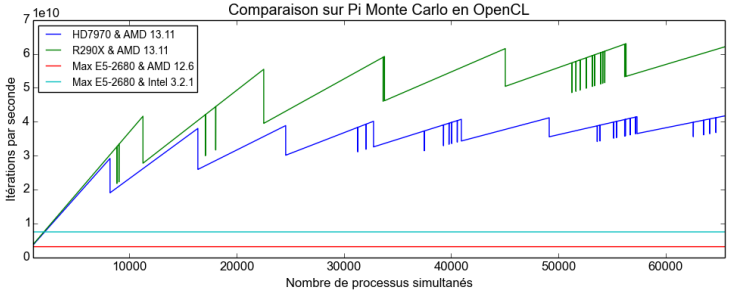
Little comparison of *PU
Low parallel rate : the reign of CPU



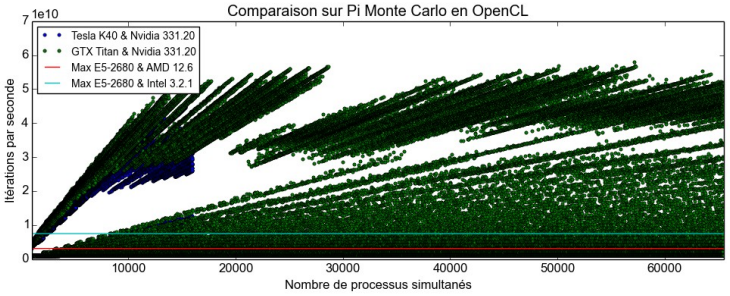
Investigate the internal structure By the execution Pi Dart Dash



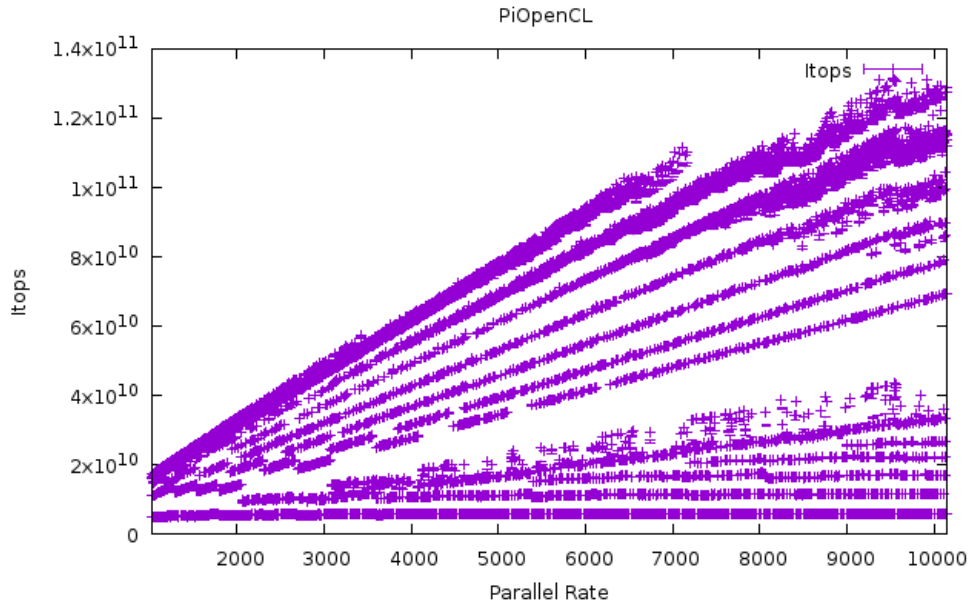
AMD HD7970 & R9-290
Long period : 4x number of ALU

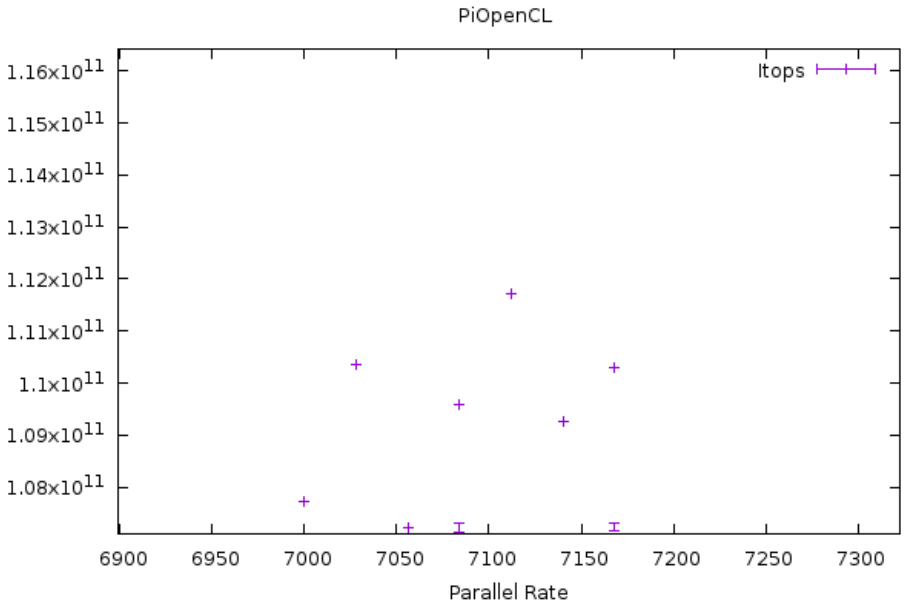


Nvidia GTX Titan & Tesla K40
Small Period : numbe of SMX unit



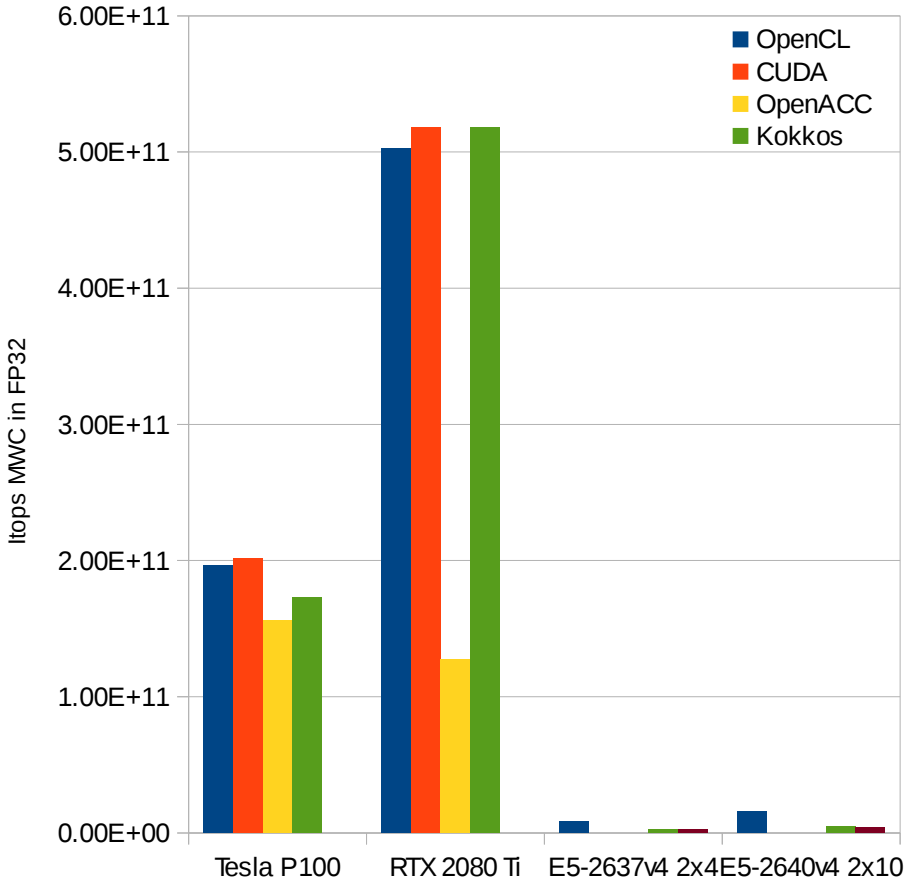
And if we use a OpenCL code written in C only ?

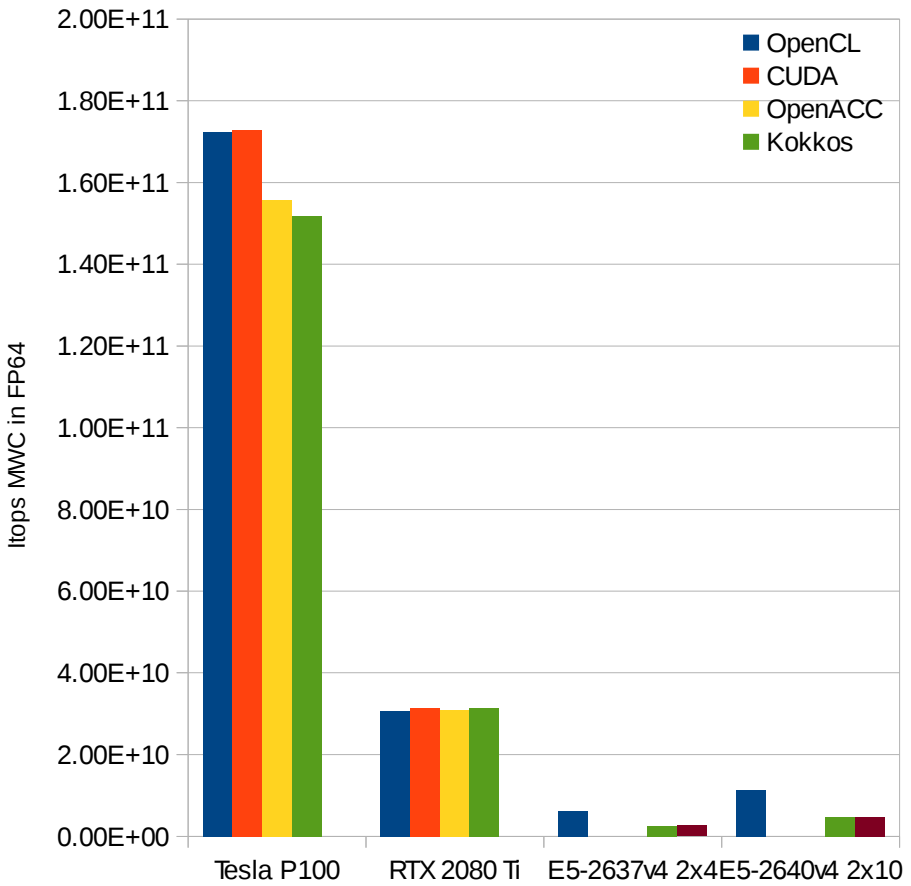




- Same detection of prime numbers
- Same local maximum (xN the number of cudacores)

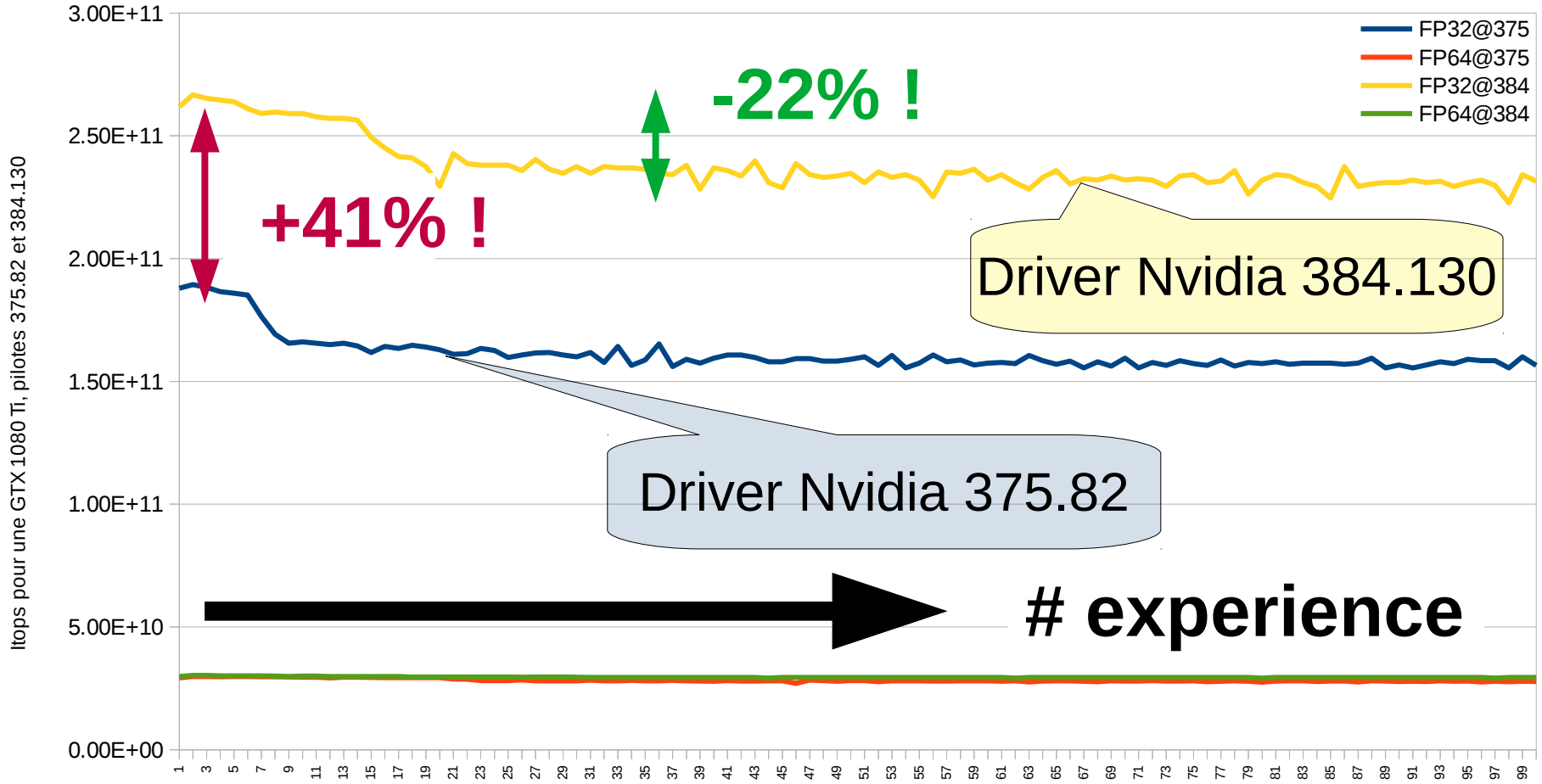
And for the other ways for GPU CUDA, OpenACC & Kokkos ?





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

Versions of drivers & experiences Variability factors...

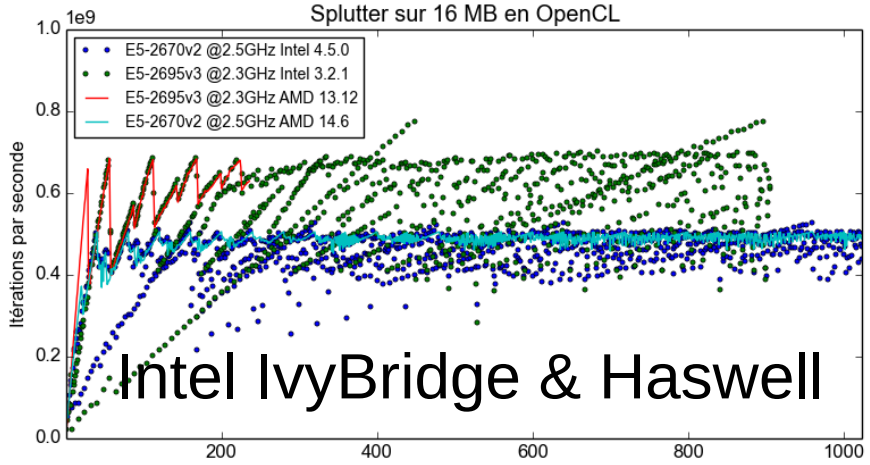


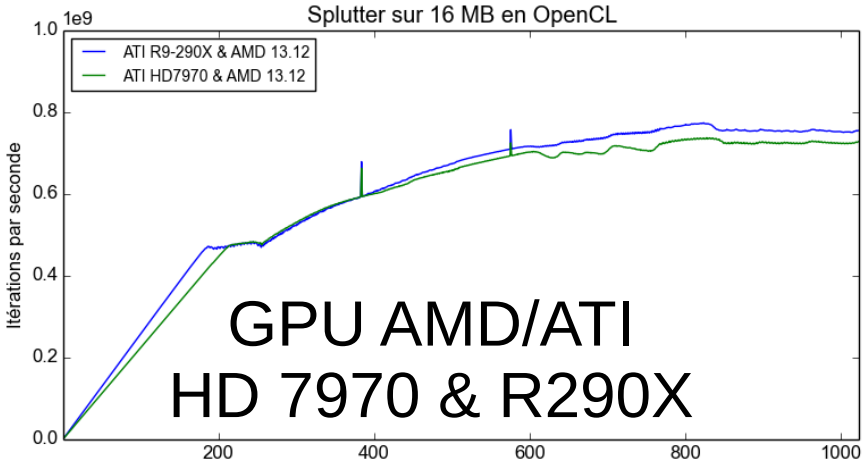
So retreive as many informations as possible...

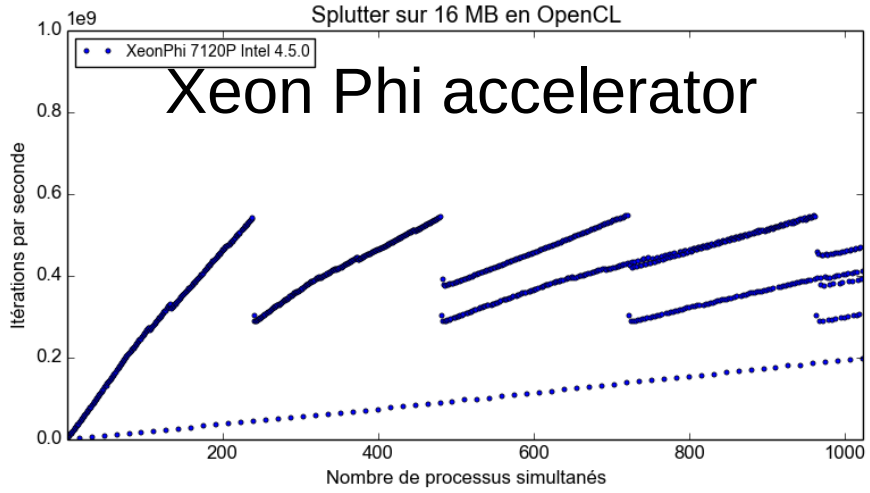
« Memory Bound » Test
The « splutter »...

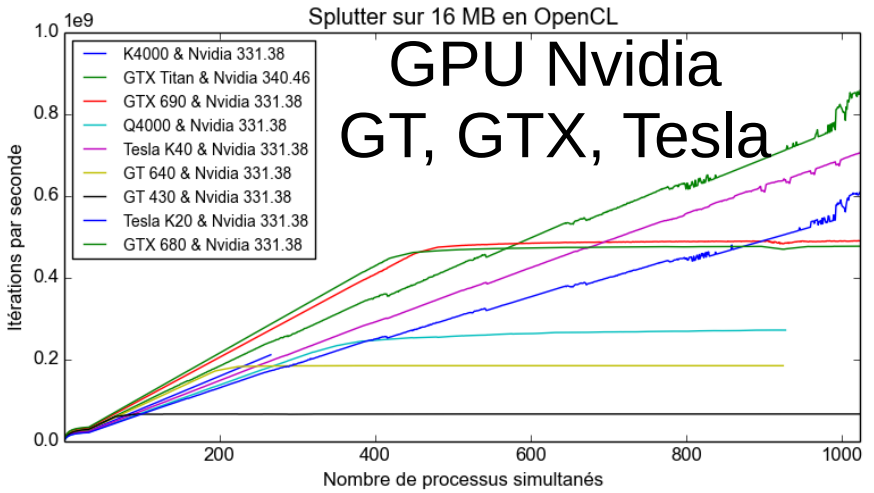
- The code :
  - I reserve a space memory
  - I transfer it on the device (GPU or CPU) : H2D
  - For each parallel task, same number of iterations
    - I take a random number modulo the reserved memory space
    - I increment with the function atomic_inc this memory cell
  - I retreive the memory space from the device : D2H
- Verification : I add all elements of memory space
  - I must find the total number of iteractions
- Observables : the 3 times…
  - Time for H2D, time for splutting, time for D2H

The « splutter » in OpenCL 5 years ago...

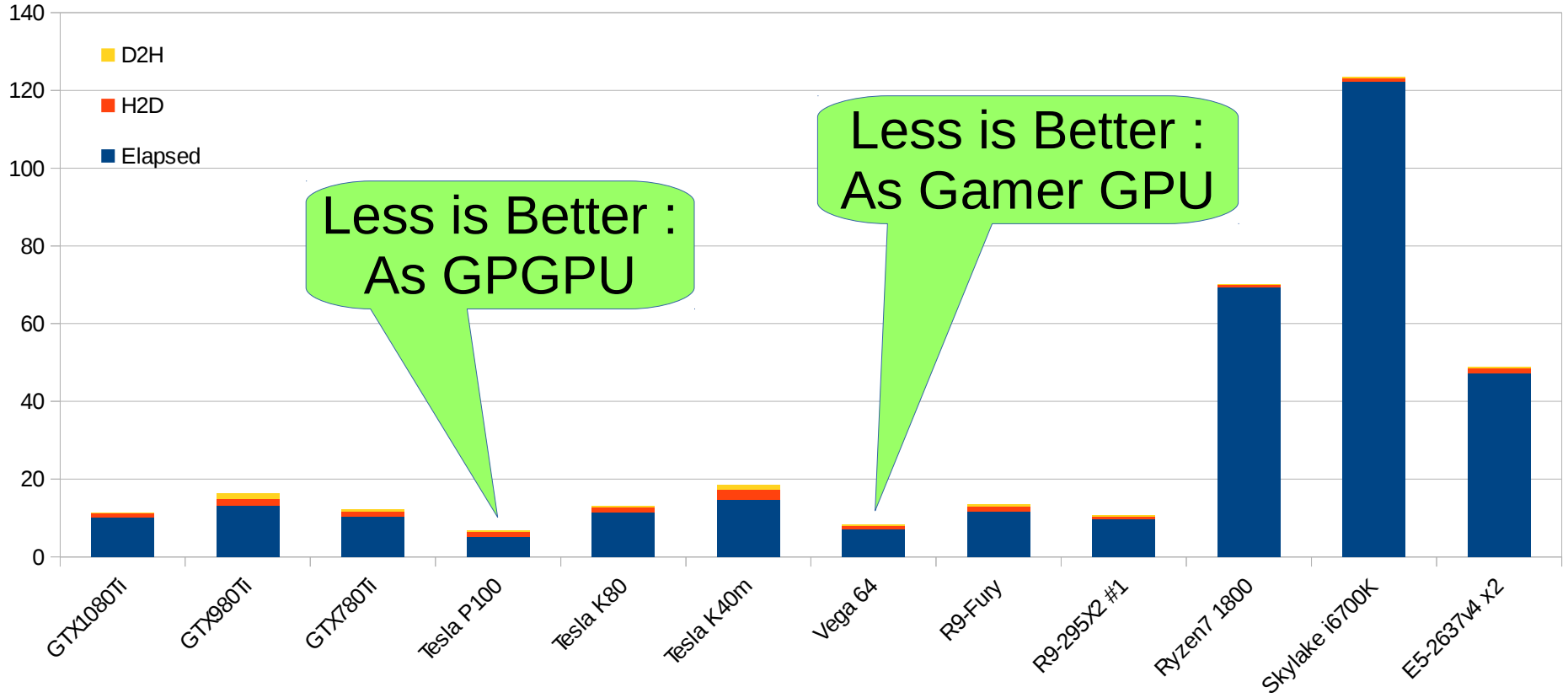








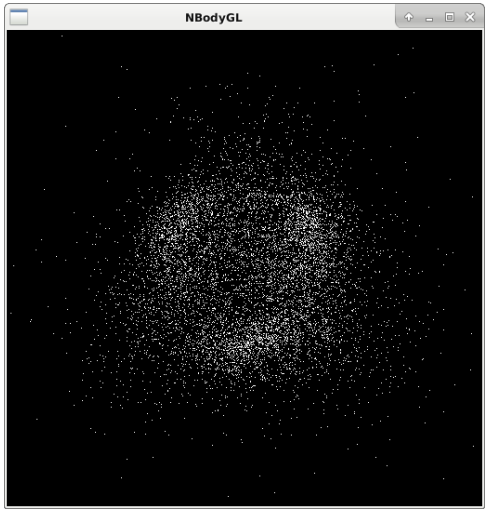
And for the GPU & CPU used ? The 9 GPU + the 3 best CPU

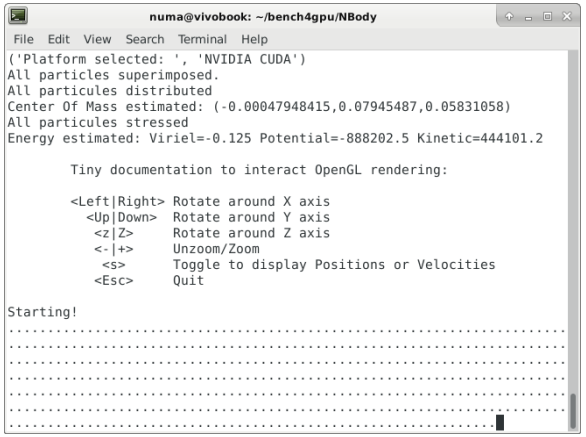


- A space of 2GB (2^{29} in 32 bits), 32768 tasks
- The memory of GPU : much more speed but size...

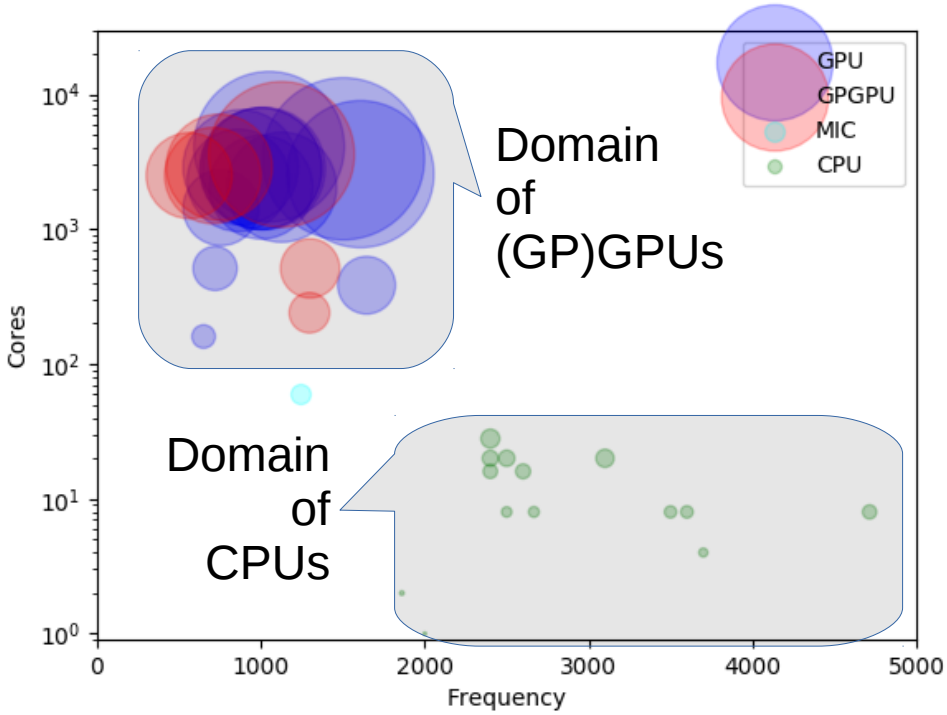
Back to physics
A newtonian N-Body code...

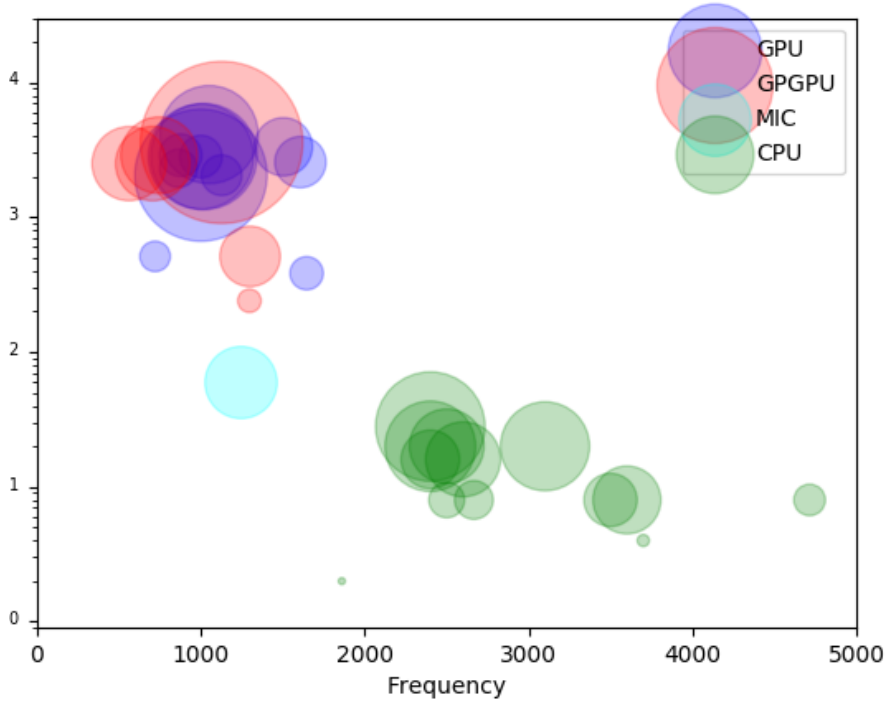
- Let's use « fine grain » code
- Code N-body very simply implemented
  - Second Newton's law, autogravitating system
- Differential integration methods :
  - Implicit Euler, Explicit Euler, Heun, Runge Kutta
- Input parameters :
  - Physical ones : number of particules
  - Numerical ones : integration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





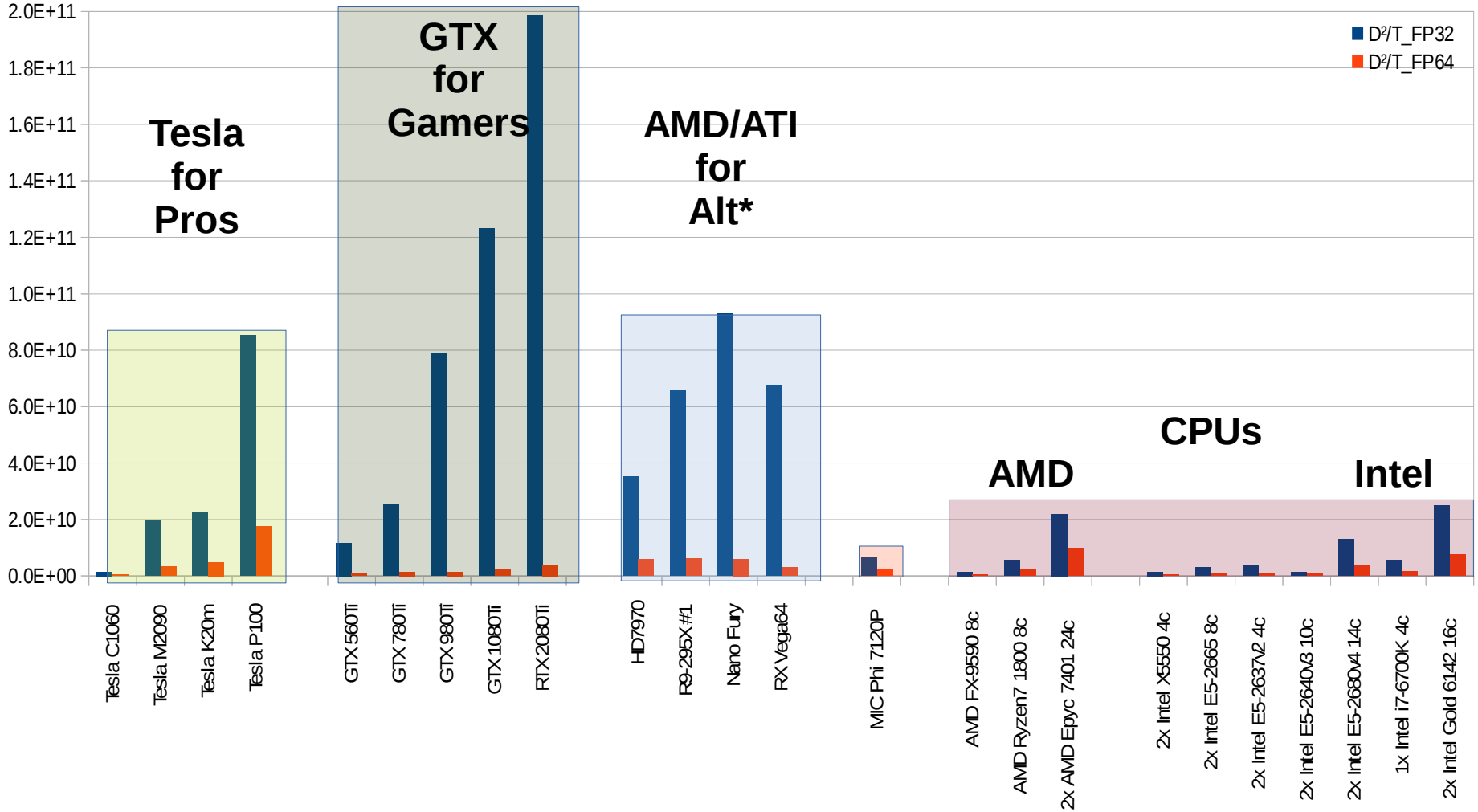
Performance on IT
A more physical code...



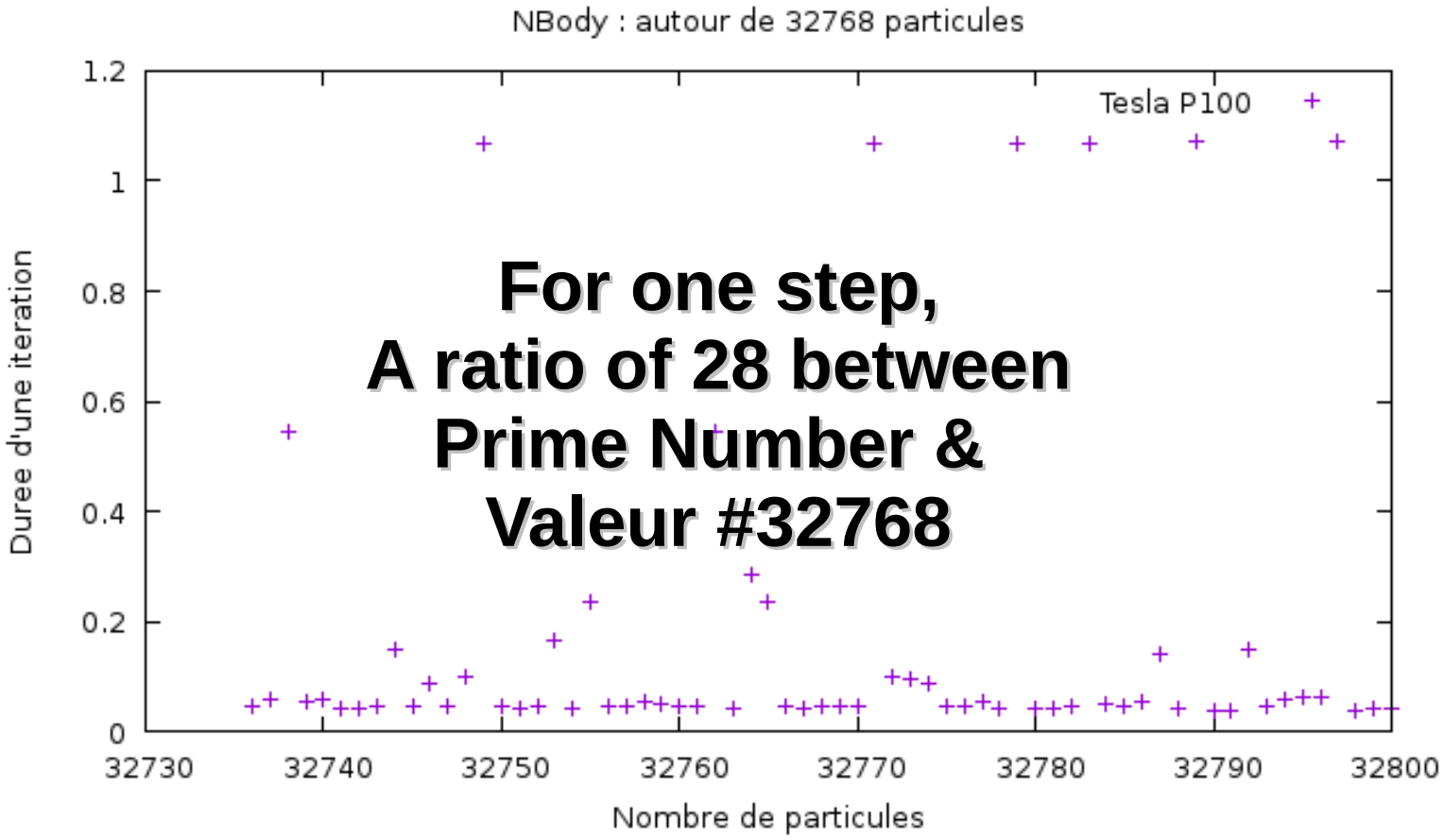


- On a GPGPU, stable performance
- On a GPU, drastic desease of performance (division by 20 in DP)
- On a CPU, performance relatively better

A « naive » N-Body code Comparison with the best...



And the « prime numbers » effect for Nvidia GPU ?



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

In conclusion
A good benchmark is : SAROS

- **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
- **Reproductible** : 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/

But why all these efforts ? Characterise & avoid the remors...

- What you must keep in mind :
  - In a machine, in 2019, the raw power resides inside the (big) GPU
  - For a low parallel rate, the CPU is better than the GPU
  - The GPU is better than the CPU only for a parallel rate greater than 1000
  - The GPU provides it’s optimal power only for specific parallel rates.
  - Without characterisation (and a relevant benchmark), deception will occur !
  - OpenCL is a good choice to make the most agnostic benchmark
  - Python stays the best way to learn the programming of GPU
- What to do : experiment !