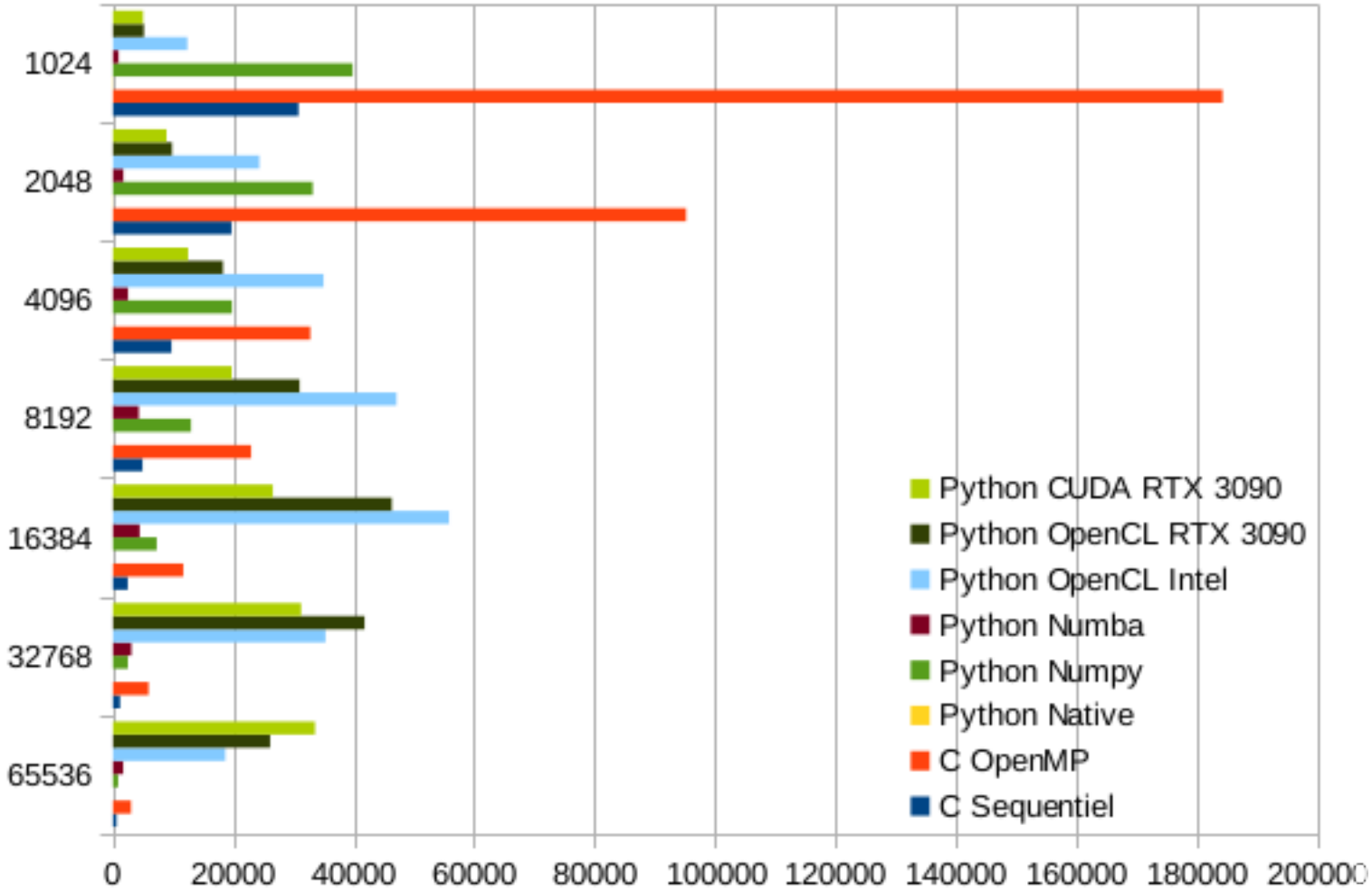
ETSN 2022

Des applications en physique en particulier
À la mesure de performance en général…
(oui... “Encore” de la métrologie...)

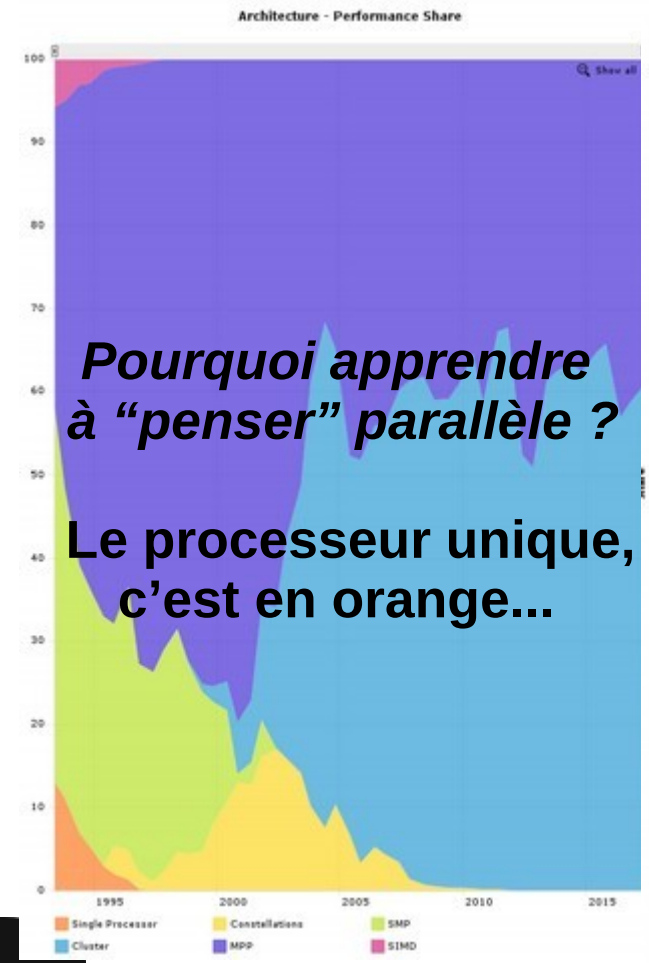
“Tout ce que nous avons à décider,
c’est quoi faire du temps qui nous est imparti”
Gandalf

Emmanuel Quémener

Implémentation « naïve » de la DFT
Des enseignements à la pelle...

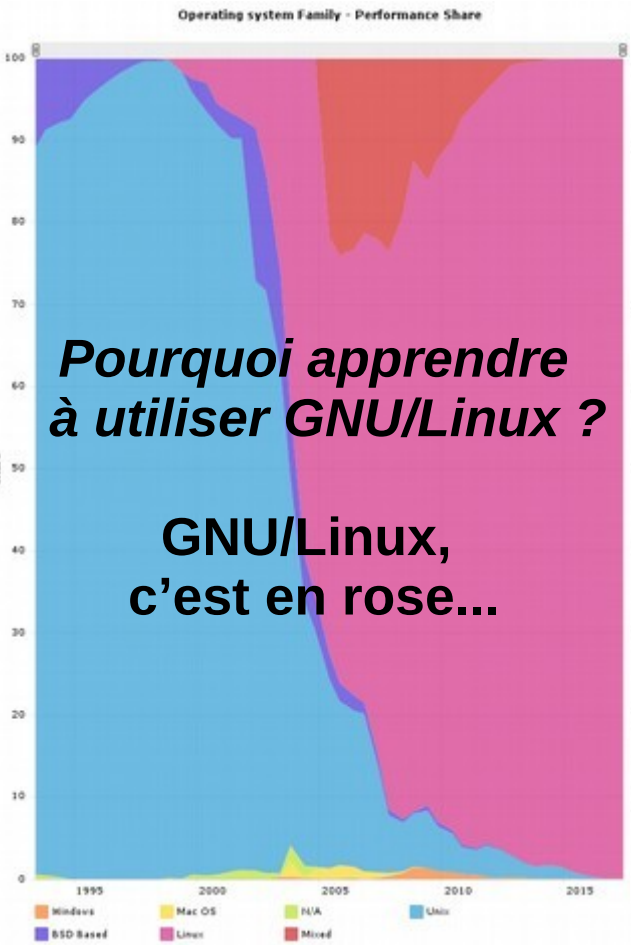


Le contexte : le TOP 500 Architectures, OS & accélérateurs...



Pourquoi apprendre à "penser" parallèle ?

Le processeur unique, c'est en orange...



Pourquoi apprendre à utiliser GNU/Linux ?

GNU/Linux, c'est en rose...

Pourquoi apprendre à exploiter les accélérateurs ?

Près d'1/3 le sont, mais... 1/3 dans le TOP10

Et ça sert ? Un exemple d'interaction Entre CBP, laboratoire, Centre de Calcul

Évaluation
Centre d'essais

Achat GPU Adapté

Savoir-Faire SIDUS
Systèmes & Réseaux

AMD

Nvidia

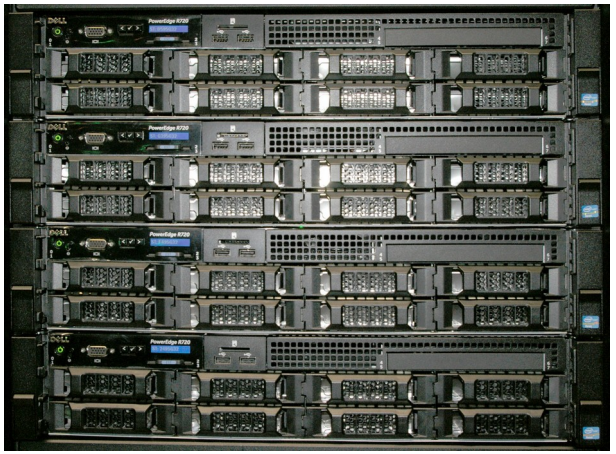
Matériel **CBP** / Sidus OS **CBP**

Pré-production
Laboratoire

Matériel **LBMC** / Sidus OS **CBP**

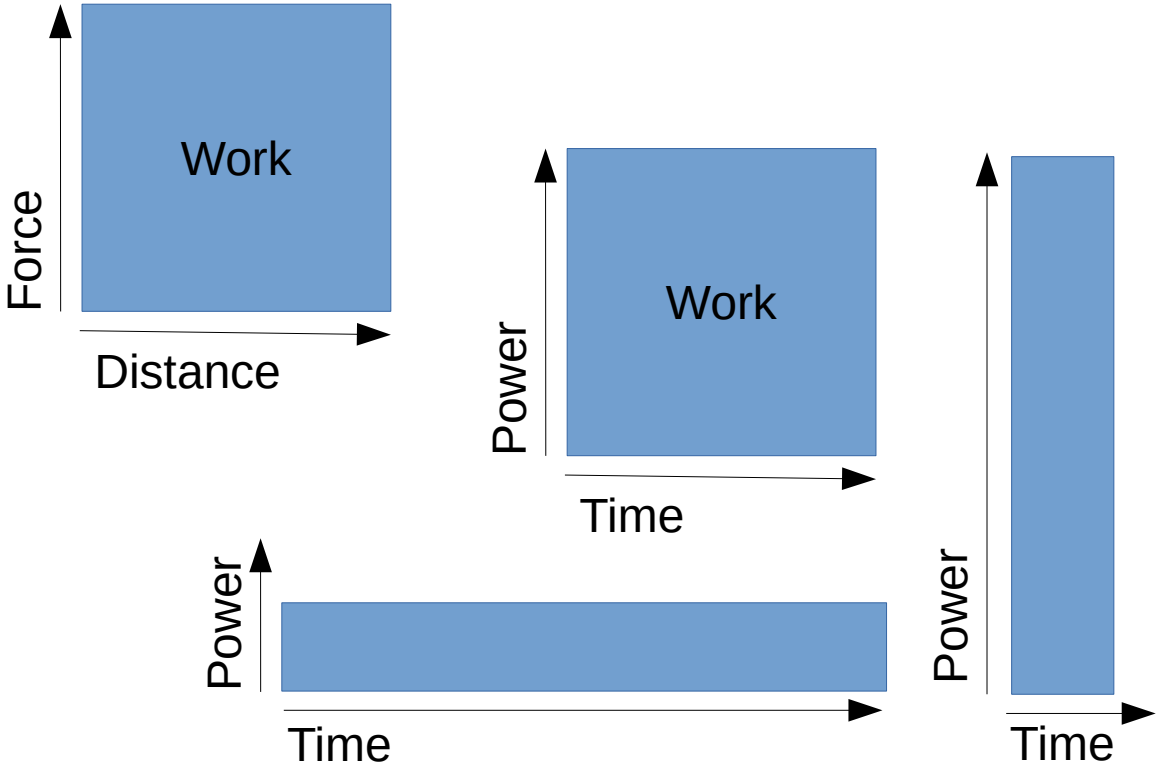
Influence de l'achat du socle matériel

Production
Centre de Calcul



Matériel **PSMN** / Sidus OS **PSMN**

« Energie » en calcul scientifique… Le point du vue du physicien



Mécanique

La puissance est définie
Comme le produit de :

- Fréquence
- Nombre d'unités
- Puissance unitaire

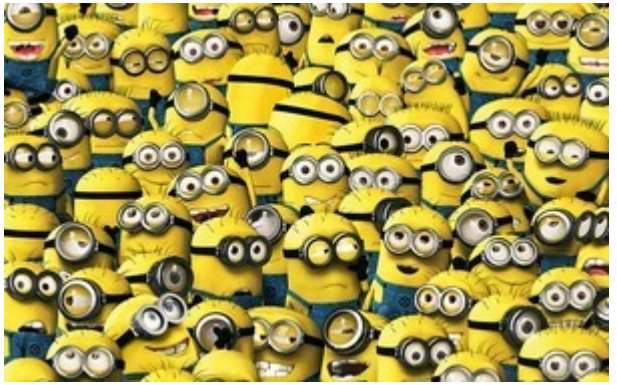
$$W = \int_{x_1}^{x_2} F\,dx = \int_{t_1}^{t_2} P\,dt$$

Caractérisation des « moteurs » de traitement à x0000 unités...

Des (multi|many)-cœurs aux myri-alus ?
Un laboratoire de //isme sur puce

- Vous avez de 1 à plusieurs dizaines de milliers de EPU !
- Vous avec le choix entre : **CPU** **GPU**

ALU **vectorielle**



Comment les choisir, les distinguer pour les exploiter ?

Qu'est-ce donc que le Code ? Un protocole d'expérimentation !

- Dans la cuisine :
  - Nous avons les ingrédients, mais nous voulons un plat !
- Dans le domaine scientifique, 3 formes :
  - Simulation : « Au service (discret?) de la théorie »
  - Traitement : pour expérimentateurs « exigeants »
  - Visualisation : voir (les choses) pour percevoir (leurs interactions) (et aussi partager !)
- Chaque exécution est une expérience (et une unique!)
  - Recettes : « codes » devenant des « processus »
  - Ustensiles : librairies, OS, matériels, réseaux, ...
  - Ingrédients : modélisation, données
  - Exécution : et une expérience NE peut se réduire à ses résultats !

Les familles de programmes

- Comment distinguer les différents codes que j'utilise ?
  - « Mon code à moi dont je suis fier ! »
  - Le code de mon chef
    - ou plutôt une stratification produite par des générations successives d'étudiants
  - Un code « métier »
    - Modèle Ikea : distribué avec des instructions de compilation
    - Modèle Crozatier : prêt à l'emploi
- Comme dans chaque famille, les problèmes avec l'héritage !
  - Dépendances avec :
    - Des librairies génériques : BLAS, Lapack, FFTw
    - Des librairies propriétaires : Mathworks, Intel, Nvidia, AMD, …
    - Le matériel !

La performance, c’est quoi ?

- D’où ça vient ce mot ?
  - Cela vient de l’anglais : « accomplir » ou « réaliser »
  - C’est également : « manière de se comporter »
  - Mais aussi : « ensemble des possibilités optimales d’un appareil »
- La « performance » :
  - c’est d’abord « faire le job »
  - c’est aussi « le faire suivant des critères (à définir) »
  - c’est enfin « le faire le mieux possible », mais quel mieux ?

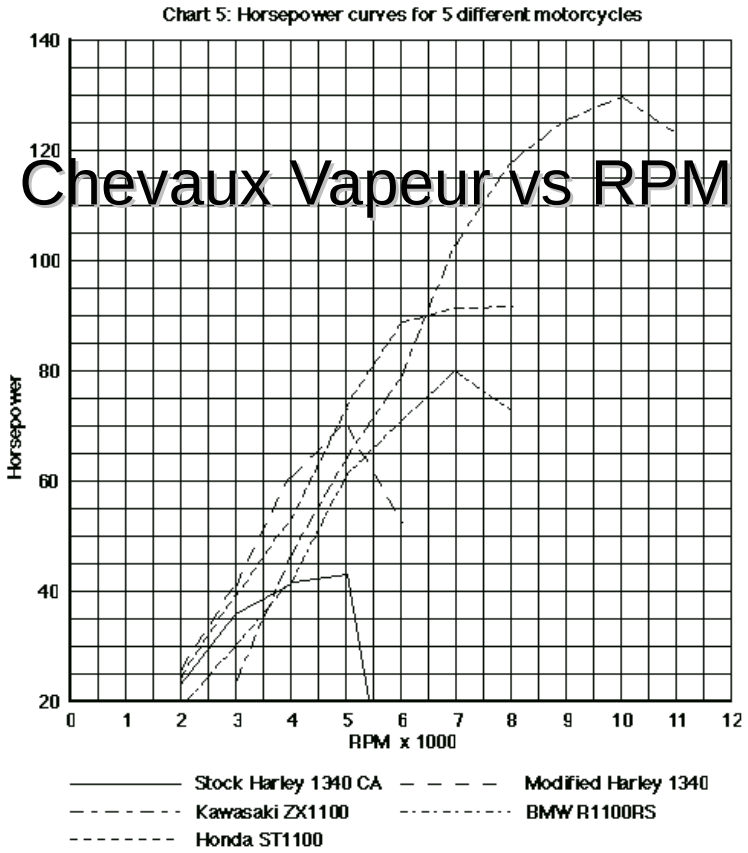
Performance : Conditionnée par les objectifs

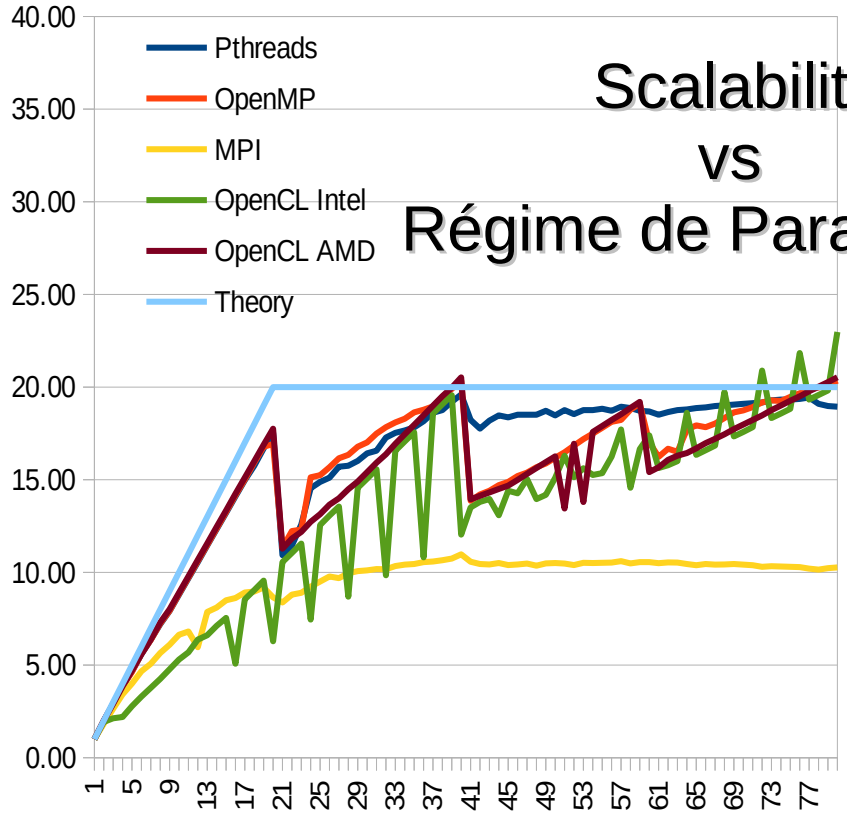
- La vitesse : Le temps écoulé (ou *Elapsed*) (mais seulement ?)
- Travail : immobilisation de ressources
- Efficacité: exploitation optimale des ressources
- Scalabilité : capacité à passer à une échelle supérieure
- Portabilité : intégration à d'autres infrastructures informatiques
- Maintenabilité : temps humain passé à maintenir le système
- Approche générale :
  - Définir un critère
  - Rechercher les valeurs extrémales sur un ensemble de tests pertinent

La vitesse comme critère « Speed, I'm Speed... »

- Toutes les durées, et pas seulement le temps d'exécution
- Dans l'utilisation d'un code, les 3 coûts :
  - Coût d'entrée : apprendre à l'utiliser, l'intégrer à l'infrastructure, …
  - Coût d'exploitation : le maintenir, l'utiliser
  - Coût de sortie : le remplacer par un autre code équivalent, ou une technologie équivalente
- Optimisation (et son biais) : DD/DE > 1 est-il pertinent ?
  - DE : Durée totale de toutes mes exécutions
  - DD : temps passé à tenter de minimiser la durée d'exécution
- Pour estimer ces valeurs :
  - Outils système, outils de métrologie dans les langages, les codes, les matériels, ...
- « Et après moi ? Le déluge ? » : quel avenir pour le code ?

Travail & Ressources Informatiques
Un moteur comme source de puissance

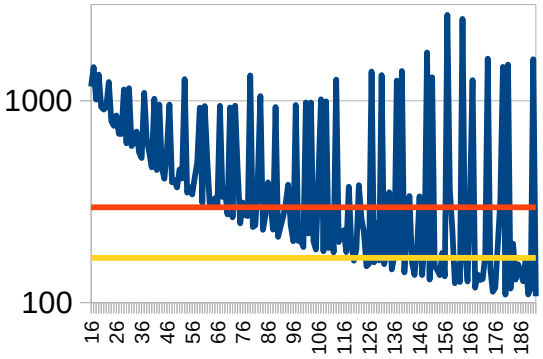


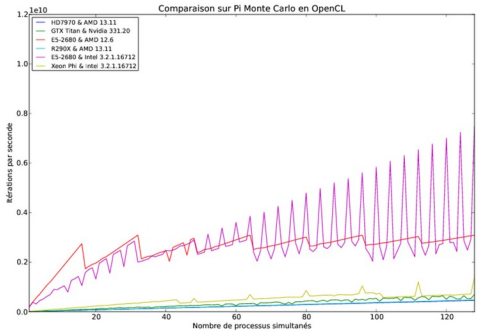


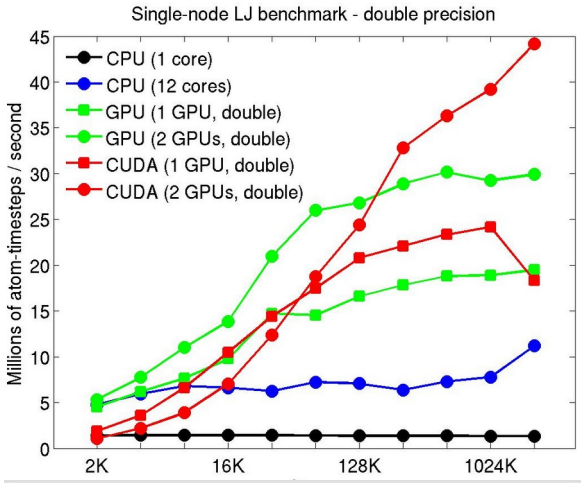
- Quel comportement de ces « moteurs » à la charge ?
- Le « moteur », un « système » englobant matériel, OS et logiciels

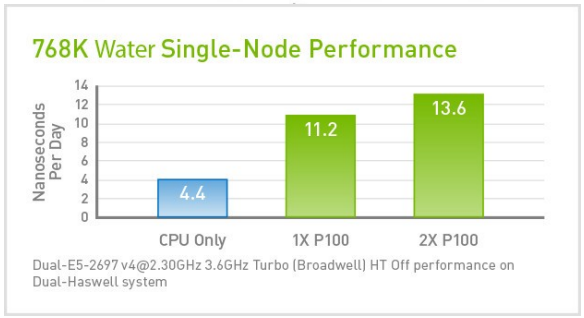
Accélération GPU ? Vérité ou mensonge

Prêt à prendre la « pilule rouge » ?





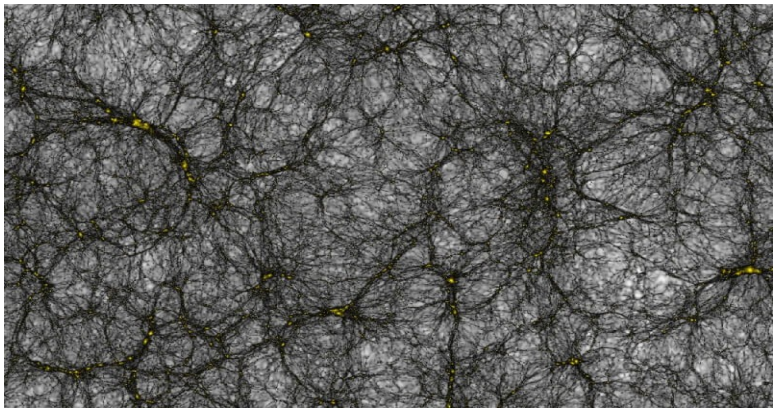




Cadre posé…
Quels codes à étudier ?

- Code « Ikea » : à compiler soi-même
  - PKDGRAV3 : astrophysique
  - Gromacs : chimie
  - D'autres...
- Code « à dépendances » :
  - Approche « intégrateur » :
    - BLAS avec xGEMM et les autres
  - Approche « développeur » :
    - Pi Dart Dash, Nbody, Splutter...

PKDGRAV3 : un client prometteur … et une bonne exposition média !



- Caractéristiques :
  - Open Source
  - Hybride : MPI, OpenMP, Cuda
- Compilation « facile »
  - Dans la doc (README) :
    - Quickstart : ./configure ; make
    - Pour le support CUDA :
      - ./configure --with-fftw --enable-integer-positions –with-cuda
      - make -j 16

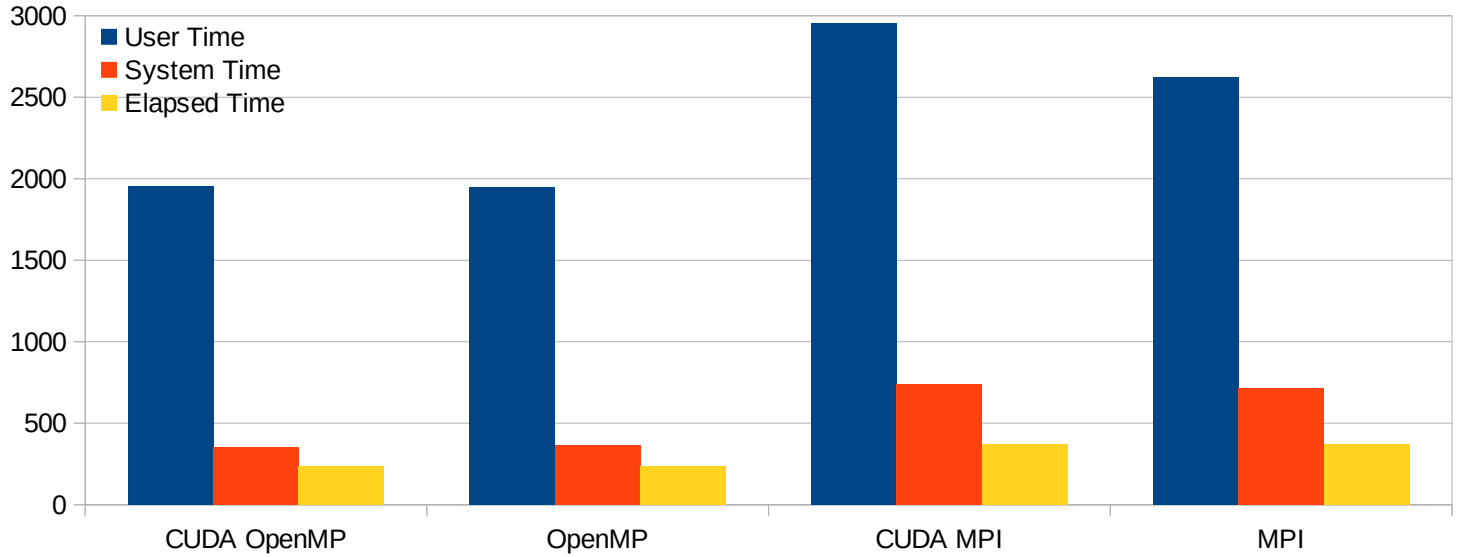
PKDGRAV3 : compilation facile ? Un peu plus compliqué sur Debian 9

- Pas de configure (donc nécessité de le générer) : ./autogen.sh
- Nécessité de compiler avec fPIC (donc nécessité de modifier) :
  - sed -i 's/compute_35/compute_35\ \-Xcompiler\ \-fPIC/g' configure.ac
- Donc en fait, compiler PKDGRAV3 avec le support CUDA, c'est :
  - cd /local ; git clone https://bitbucket.org/dpotter/pkdgrav3.git
  - PKDGRAV3=/local/pkdgrav3-$(date "+%Y%m%d") ; mv pkdgrav3 $PKDGRAV3
  - cd $PKDGRAV3
  - sed -i 's/compute_35/compute_35\ \-Xcompiler\ \-fPIC/g' configure.ac
  - ./autogen.sh
  - ./configure --with-fftw –enable-integer-positions --with-cuda
  - make -j 16

PKDGRAV3 : on veut comparer ! Donc, pour une pure « MPI »

- On ajoute aux commandes précédentes :
  - mv pkdgrav3_mpi pkdgrav3_mpi_cuda
  - make distclean-recursive
  - ./autogen.sh
  - ./configure --with-fftw --enable-integer-positions
  - make -j 16
- Et comme test ?
  - cosmology.par dans examples...

PKDGRAV3 : et les résultats ? Sur une Tesla P100 & E5-2640v4



- Aucun gain !
- Tout se passe comme si le GPU était absent…
- Mais OpenMPI et le programme accèdent au périphérique…
- Bref, j'ai dû merdé quelque part mais je ne sais pas où…
  - En fait, il y avait une option à activer au lancement mais pas dans la documentation...

Gromacs : le bon « client » de la chimie

- Socle matériel : les plates-formes du CBP
  - 51 GPU différents, 20 types de CPU
- Socle logiciel : Debian 9.4 dite « stretch » en AMD64
- Caractéristiques :
  - Open Source
  - Hybride : multi-nœuds, multi-cœurs, multi-shaders
  - Possibilité d'exécution CPU (MPI+OpenMP), CPU+GPU
  - Paramétrage facile du parallélisme et du choix du GPU

Gromacs : phase préparatoire
En lisant la documentation

- Expansion archive, déplacement, création dossier, déplacement
  - tar xfz gromacs-2018.1.tar.gz ; cd gromacs-2018.1 ; mkdir build ; cd build
- Paramétrage compilation avec options :
  - cmake .. -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON
- Compilation et installation
  - make ; make check ; sudo make install
- Chargement de l'environnement :
  - source /usr/local/gromacs/bin/GMXRC

Gromacs : phase préparatoire
Nos contraintes & nos 1ers déboires

- Plusieurs contextes : à chaque processeur/GPU, sa compilation
  - Plusieurs dossiers d'installation des exécutables
- Compilateurs standards Stretch : gcc, g++ version 6.3
- Lancement cmake : KO

```
CMake Error at cmake/gmxManageGPU.cmake:289 (message):
  NVCC/C compiler combination does not seem to be supported.  CUDA frequently
  does not support the latest versions of the host compiler, so you might
  want to try an earlier C/C++ compiler version and make sure your CUDA
  compiler and driver are as recent as possible.
Call Stack (most recent call first):
  CMakeLists.txt:582 (gmx_gpu_setup)

-- Configuring incomplete, errors occurred!
See also "/local/Gromacs/gromacs-2018.1/build_CUDA/CMakeFiles/CMakeOutput.log".
  See also "/local/Gromacs/gromacs-2018.1/build_CUDA/CMakeFiles/CMakeError.log".
```

- Trouver un autre compilateur !

Gromacs : phase préparatoire Les déboires continuent

- Passage sur clang version 3.8, mais…

```
-- Adding work-around for issue compiling CUDA code with glibc 2.23 string.h
CMake Warning at cmake/gmxManageGPU.cmake:272 (message):
  To use GPU acceleration efficiently, mdrun requires OpenMP multi-threading.
  Without OpenMP a single CPU core can be used with a GPU which is not
  optimal.  Note that with MPI multiple processes can be forced to use a
  single GPU, but this is typically inefficient.  You need to set both C and
  C++ compilers that support OpenMP (CC and CXX environment variables,
  respectively) when using GPUs.
Call Stack (most recent call first):
  CMakeLists.txt:582 (gmx_gpu_setup)
```

  - Exécution dégradée (pas de OpenMP)
  - Phases cmake & compilation effroyablement lente (plus de 10x)
  - Finalement crash à la compilation… :-(

- Passage sur compilateur plus ancien 4.9

Gromacs : phase préparatoire Importation d'un vieux compilateur

- Exploitation des « snapshots » : tout le script d'installation...
  - wget http://snapshot.debian.org/package/gcc-4.9/4.9.4-2/
  - mkdir gcc-4.9.4
  - cd gcc-4.9.4
  - wget -O DebianSnapshot-gcc-4.9.4.html http://snapshot.debian.org/package/gcc-4.9/4.9.4-2/
  - egrep '(_amd64.deb|_all.deb)' DebianSnapshot-gcc-4.9.4.html | awk -F\" '{ print $2 }' | grep -v kfreebsd | grep -v lib32 | grep -v libx32 | sort -u | grep -v multilib | xargs -I '{}' wget http://snapshot.debian.org/'{}'
  - sudo dpkg -i *deb
  - sudo apt-get -f install
- Ca y est ! Un vieux compilateur prêt pour la route...

Gromacs : enfin la compilation ! Tout le script, pour un CUDA

- mkdir -p /local/Gromacs/ ; cd /local/Gromacs/
- wget ftp://ftp.gromacs.org/pub/gromacs/gromacs-2018.1.tar.gz
- export GPU=GTX1080Ti
- export CC=/usr/bin/gcc-4.9 ; export CXX=/usr/bin/g++-4.9
- export GMXINSTALL=/scratch/Gromacs/2018.1
- export GMXSRC=/local/Gromacs/gromacs-2018.1
- tar xzf gromacs-2018.1.tar.gz ; cd gromacs-2018.1
- mkdir -p $(dirname $GMXINSTALL) ; mkdir $GMXSRC/build_${GPU}
- cd $GMXSRC/build_${GPU}
- cmake .. -DGMX_BUILD_OWN_FFTW=ON -DGMX_GPU=on -DREGRESSIONTEST_DOWNLOAD=ON -DCMAKE_INSTALL_PREFIX=${GMXINSTALL}_${GPU}
- make -j 16 ; make check ; make install

Gromacs : cas d'usage Nvidia Paramètres d'exécution

- Tâche coûteuse (et parallélisée) : mdrun
  - -ntmpi : contrôle du nombre de processus MPI concurrent
  - -ntomp : contrôle du nombre de processus OpenMP concurrents
  - -gpu_id : contrôle l'exécution sur un (ou plusieurs) GPU
  - -nb cpu : contrôle l'exécution sur uniquement le CPU
- Si aucun paramètre donné : mdrun explore la machine
  - Autant de processus OpenMP que de cœurs détectés (si aucun OMP_NUM_THREADS)
  - Autant de processus MPI que de cœurs détectés (si OMP_NUM_THREADS mis à 1)
- Dans une exécution multigpu : -gpu_id 01 active le 0 et le 1
  - Pour les Nvidia, préférez la variable d'environnement : CUDA_VISIBLE_DEVICES

Le banc de test... D'abord les 9 (GP)GPUs

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

Le banc de Test : Ensuite leur socle, les 7 CPU

- Ryzen7 1800 : 8 cœurs, 16HT, 3.6 GHz
  - Socle du R9 Fury
- Skylake i7-6700K : 4 cœurs, 8HT, 4 GHz
  - Socle du Vega64 et du R9-295X2
- E5-2637v4 : 2x4 cœurs, 16HT, 3.5 GHz
  - Socle du GTX1080 Ti
- E5-2640v4 : 10 cœurs virtualisés, 2.4 GHz
  - Socle du Tesla P100
- E5-2637v2 : 4 cœurs virtualisés, 3.5 GHz
  - Socle du Tesla K40m
- E5-2607v2 : 2x4 cœurs, 2.5 GHz
  - Socle du GTX980Ti
- E5- 2620 : 6 cœurs, 12 HT, 2 GHz
  - Socle du GTX 780Ti

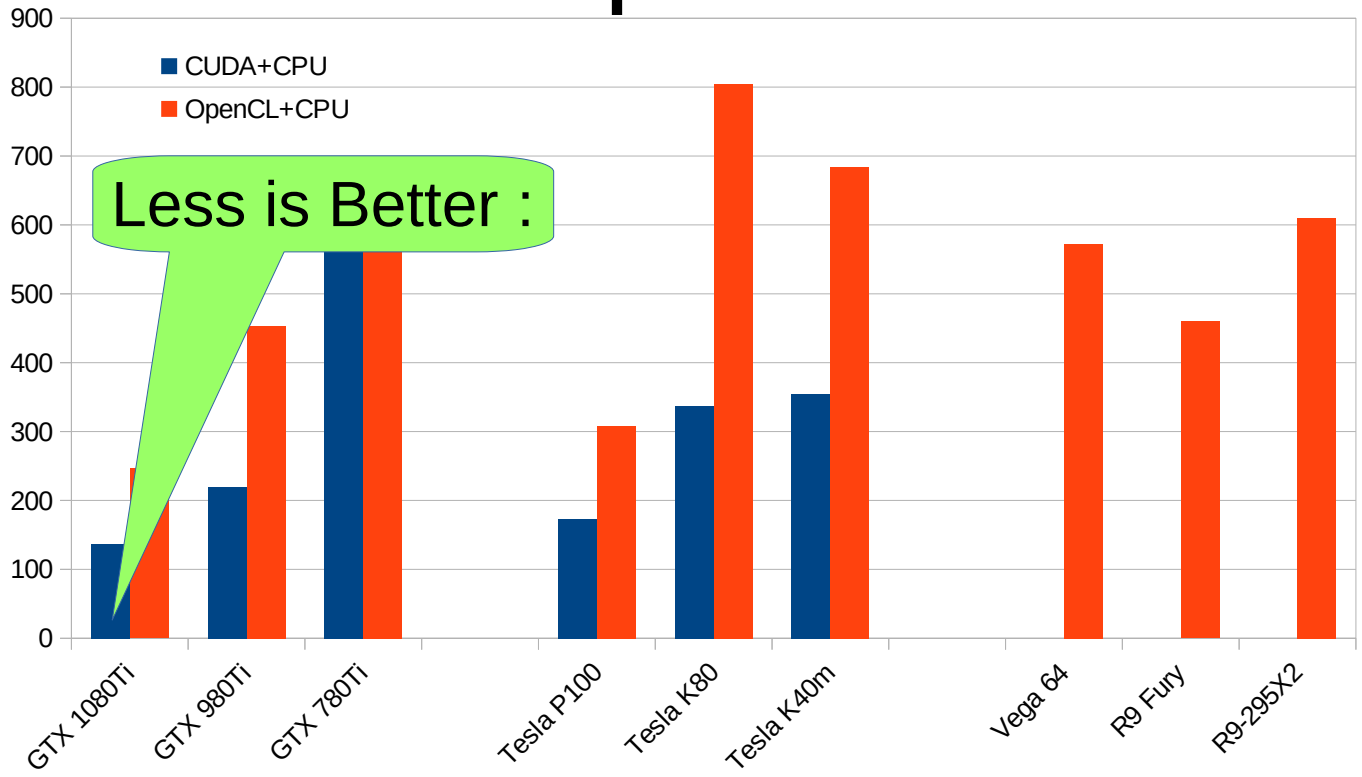
Gromacs : cas d'usage #1 « Test » Nvidia

- Configuration des chemins et dossiers d'exécution
  - 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
- Chargement & expansion de l'archive de paramètres d'entrée
  - 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
- Exécution des tâches
  - $GRODIR/bin/gmx grompp -f pme.mdp
  - /usr/bin/time $GRODIR/bin/gmx mdrun -resethway -noconfout -nsteps 4000 -v -gpu_id 0

Gromacs : cas d'usage #1
« Test » Nvidia : sortie « time »

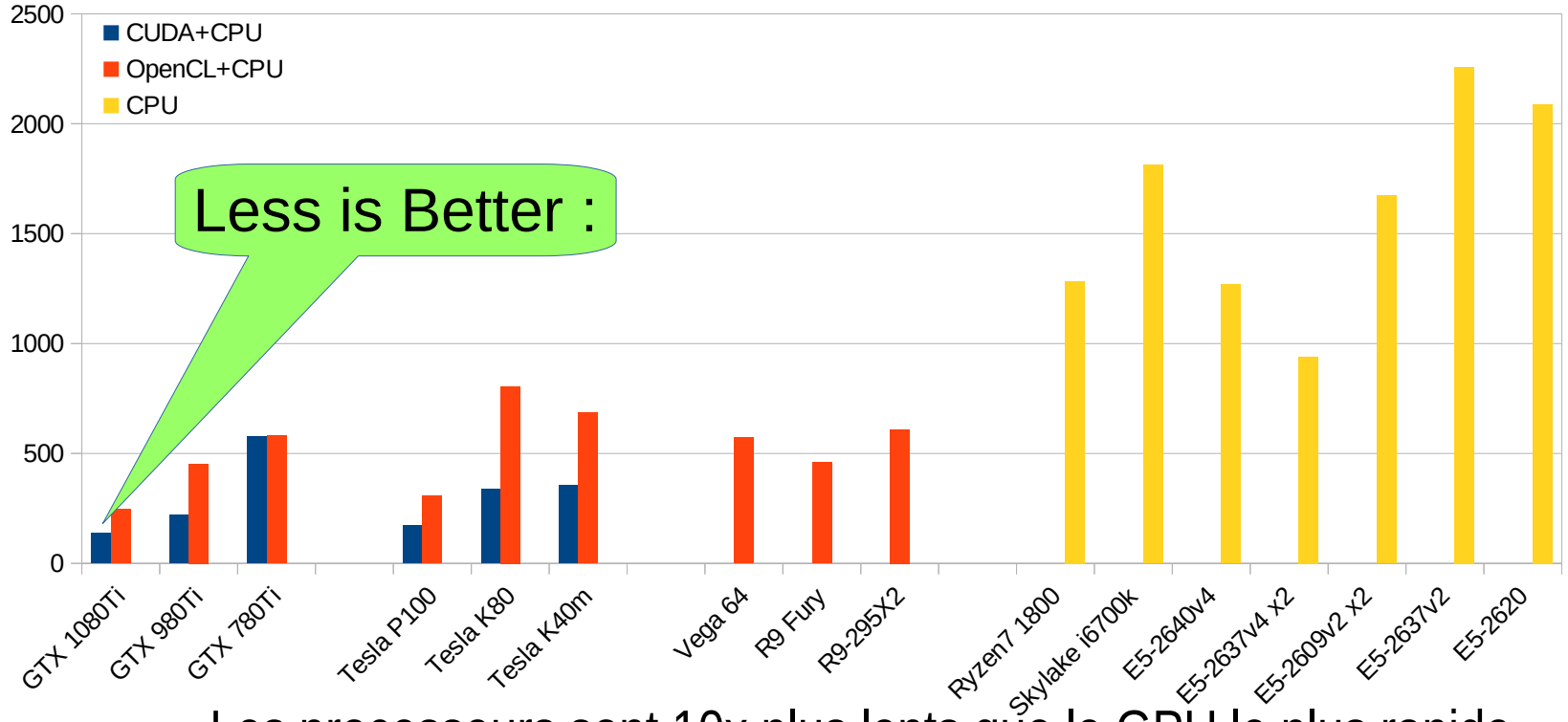
- TIME Command being timed: "/scratch/Gromacs/2018.1_GTX980Ti_OpenCL/bin/gmx mdrun -resethway -noconfout -nsteps 4000 -v -gpu_id 0"
- TIME User time (seconds): 2767.16
- TIME System time (seconds): 25.98
- TIME Elapsed (wall clock) time : 452.90
- TIME Percent of CPU this job got: 616%
- TIME Average 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

Cas d'usage #1 : test Nvidia D'abord pour les GPU...



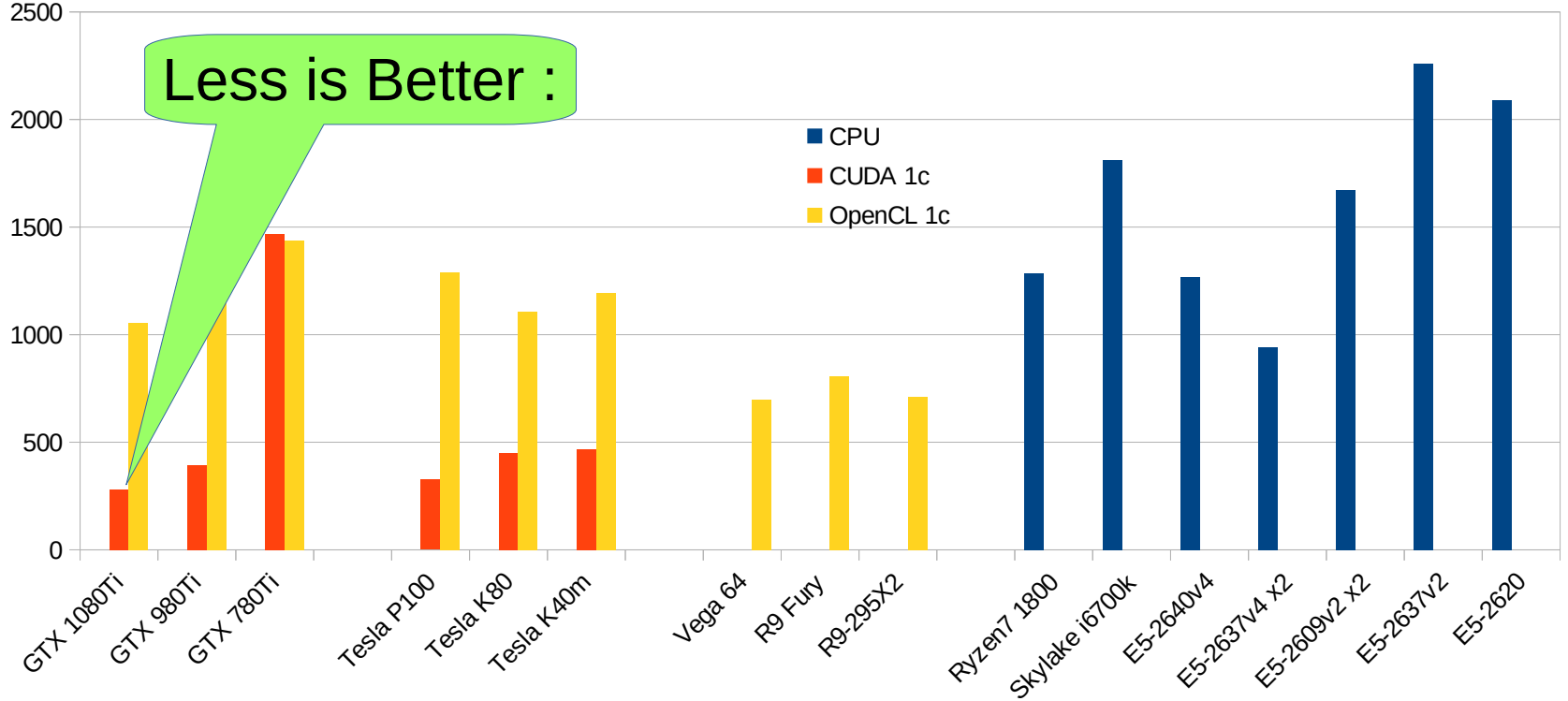
- Les implémentations OpenCL sont 2x plus lentes
- Les GPU AMD sont au moins 4x plus lentes

Cas d'usage #1 : test Nvidia
Ensuite on rajoute les CPU



  - Les processeurs sont 10x plus lents que le GPU le plus rapide
  - Les GPU sont au pire 2x plus rapide que le CPU le plus rapide
- Escroquerie ! Le programme est hybride (CPU/GPU)

Cas d'usage #1 : test Nvidia Renormaliser : GPU sur 1 cœur...



- Le GPU le plus puissant sur 1 cœur CPU est 4x plus rapide que le CPU
- Les GPU sont dans tous les cas plus rapides que les assemblages de CPU

- Donc, utiliser des GPU pour Gromacs, c'est pertinent...

Gromacs : cas d'usage #2
« Test » Tutoriel Energie libre

- Configuration des chemins et dossiers d'exécution
  - export GPU=GTX780Ti
  - export GRODIR=/scratch/Gromacs/2018.1_${GPU}
  - source $GRODIR/bin/GMXRC
  - cd /scratch/Gromacs/Test ; mkdir $(hostname)_${GPU} ; cd $(hostname)_${GPU}
- Chargement & expansion de l'archive de paramètres d'entrée
  - tar xzf ../gromacs-free-energy-tutorial.tgz ; cd gromacs-free-energy-tutorial
  - SIZE=10
- Exécution des tâches
  - $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

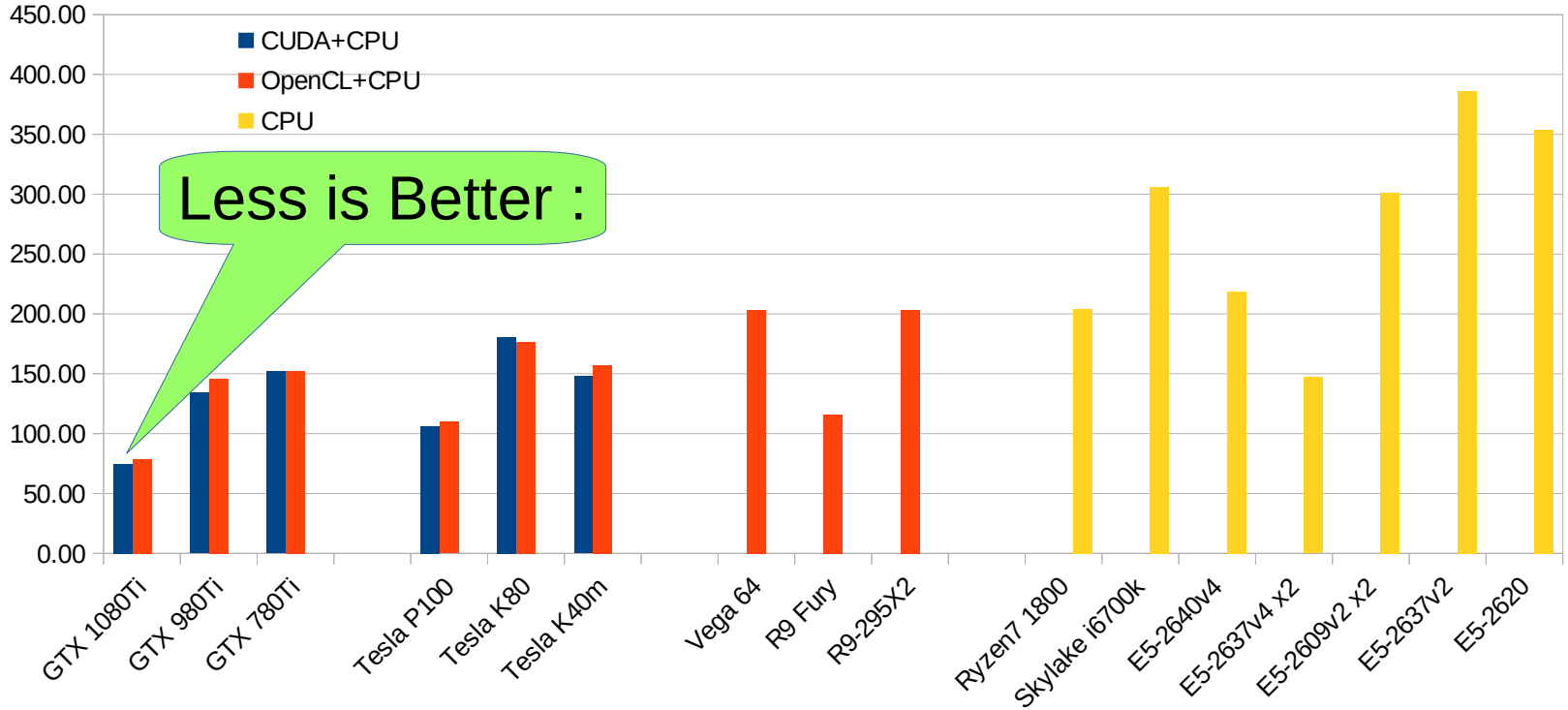
Gromacs : cas d'usage #2
« Test » Tutoriel Energie libre

- Une sortie beaucoup plus compacte :
  - TIME User time (seconds): 909.79
  - TIME System time (seconds): 4.95
  - TIME Elapsed (wall clock) time : 145.61

Cas d'usage #2 : test Tutorial D'abord pour les GPU...

- Les implémentations OpenCL équivalentes à celles CUDA
- Les GPU AMD entre 2x et 3x plus lent que la meilleure

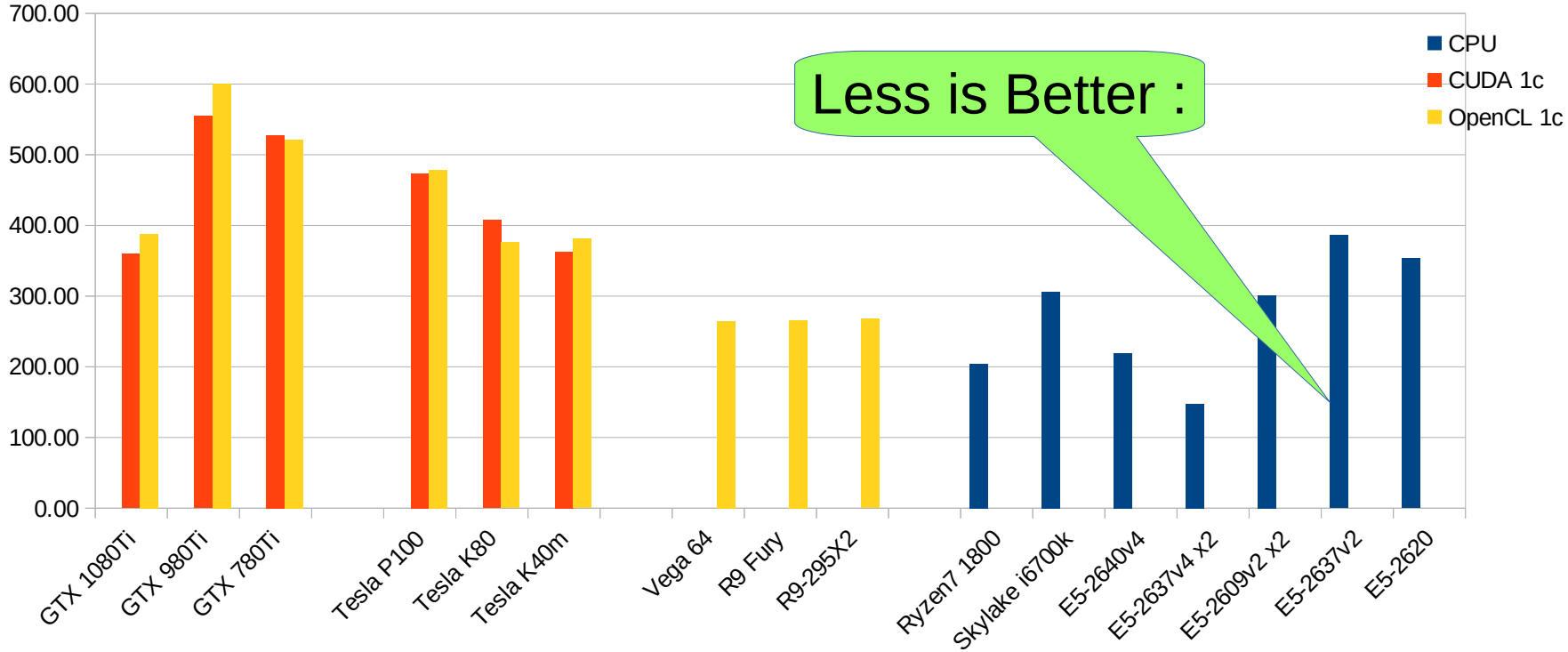
Cas d'usage #2 : test Nvidia Ensuite on rajoute les CPU



- Les processeurs ne sont plus que 6x plus lents que le GPU le plus rapide
- Les GPU les moins rapides sont moins rapide que le CPU le plus rapide

- Et toujours l'escroquerie ! Le programme est hybride (CPU/GPU)

Cas d'usage #2 : test Nvidia Renormaliser : GPU sur 1 cœur...



- Les GPU AMD sont plus rapides dans tous les cas que les Nvidia
- Les CPU sont généralement plus rapides les GPU

- Donc, utiliser des GPU pour Gromacs ici, attention !

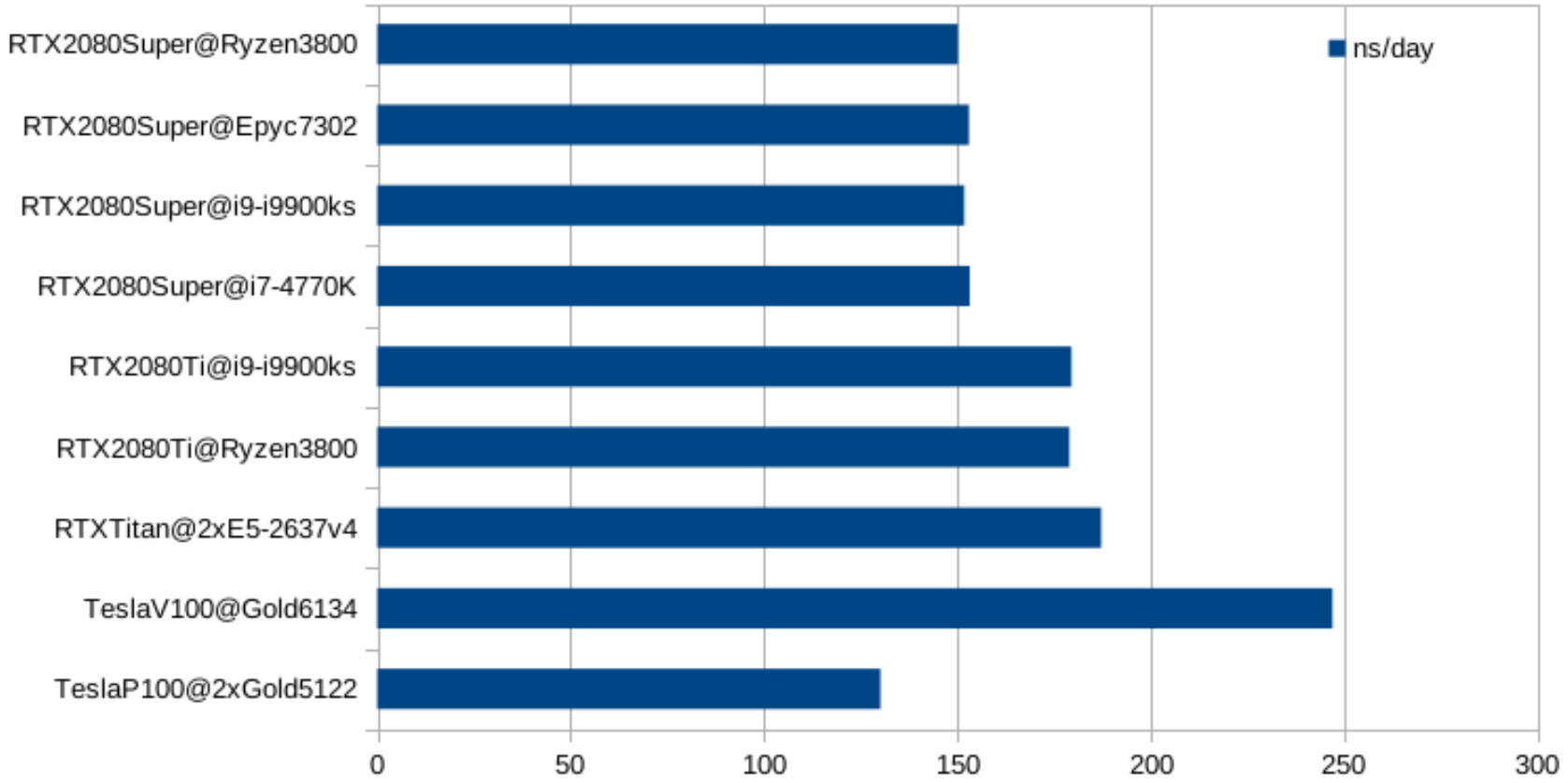
Gromacs : la conclusion

- GPU efficaces, MAIS dans un BON environnement
  - Le nombre de cœurs & la vitesse de la mémoire importants
- Implémentation CUDA toujours plus efficace
- Implémentation OpenCL en progrès
  - Mais 2x moins performante que la CUDA
- Exploitation de cartes de Gamer pertinente
  - Mais dans le cas de calculs en simple précision
- En gros, sans expérimentation, pas d'utilisation optimale…

Mais où est passé le facteur 100 de la pub ?

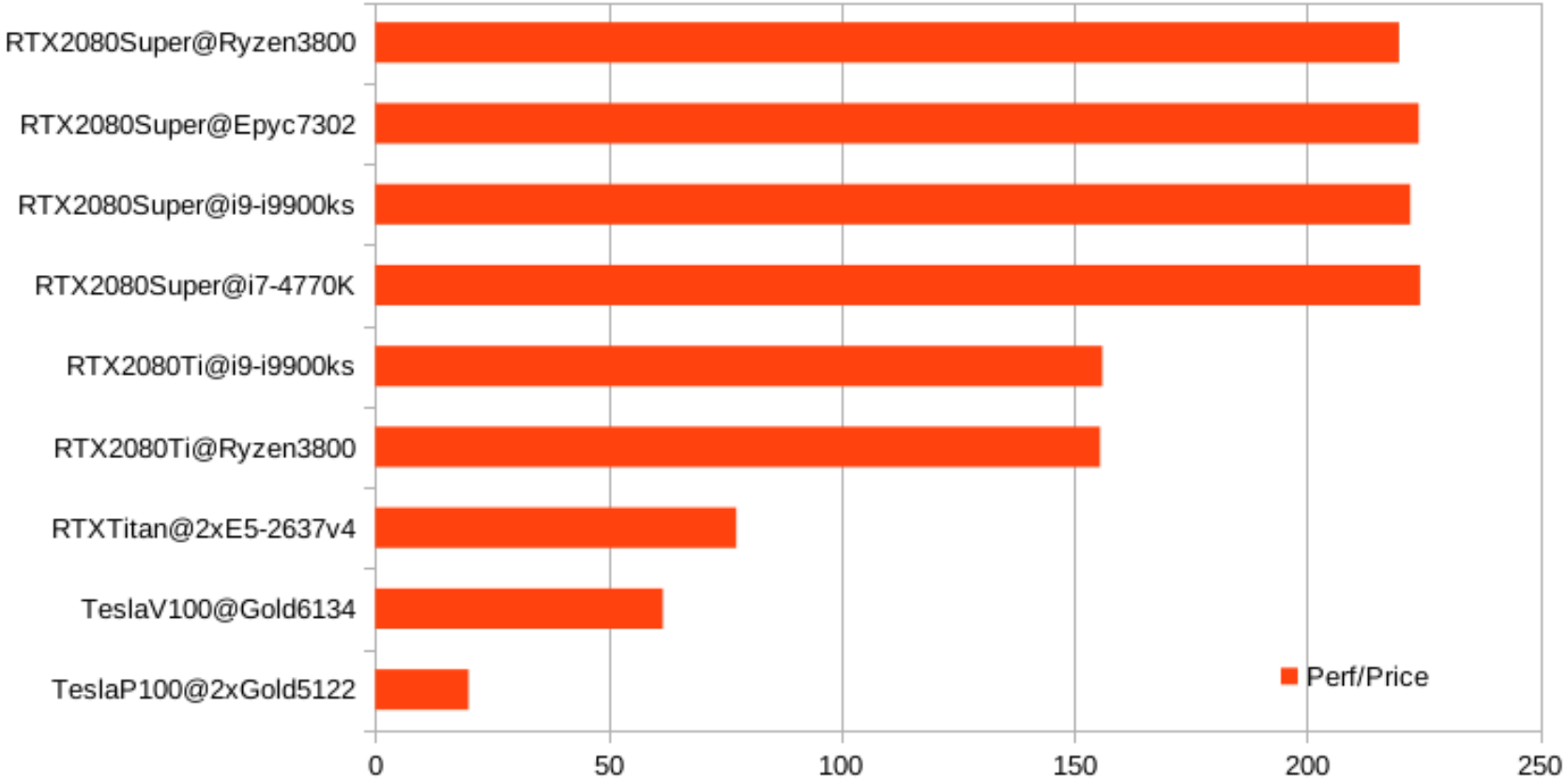
Un code « métier » propriétaire Amber9

- Si seulement la performance compte...



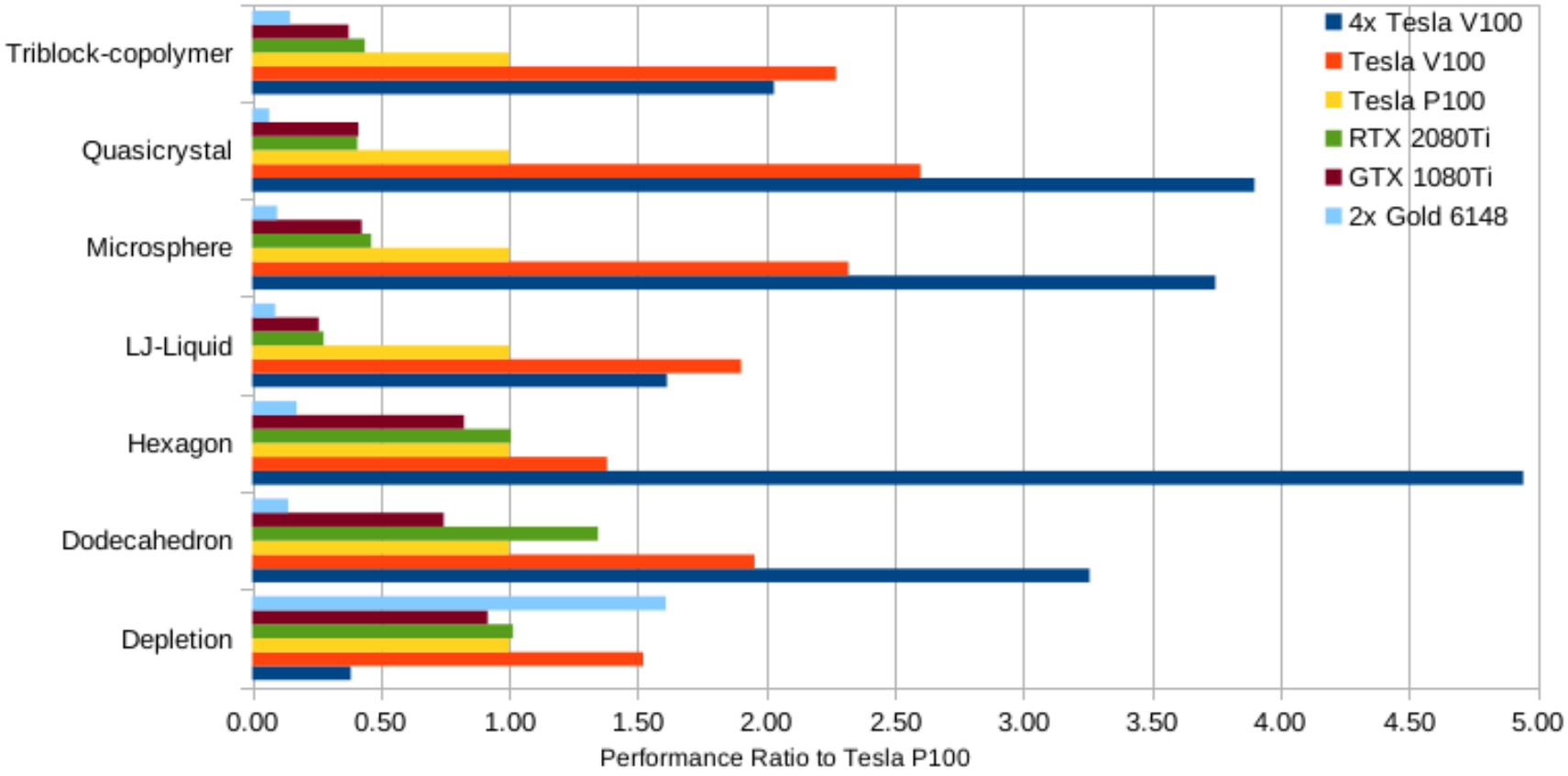
Un code « métier » propriétaire Amber9

- Si vous avez un budget limité...



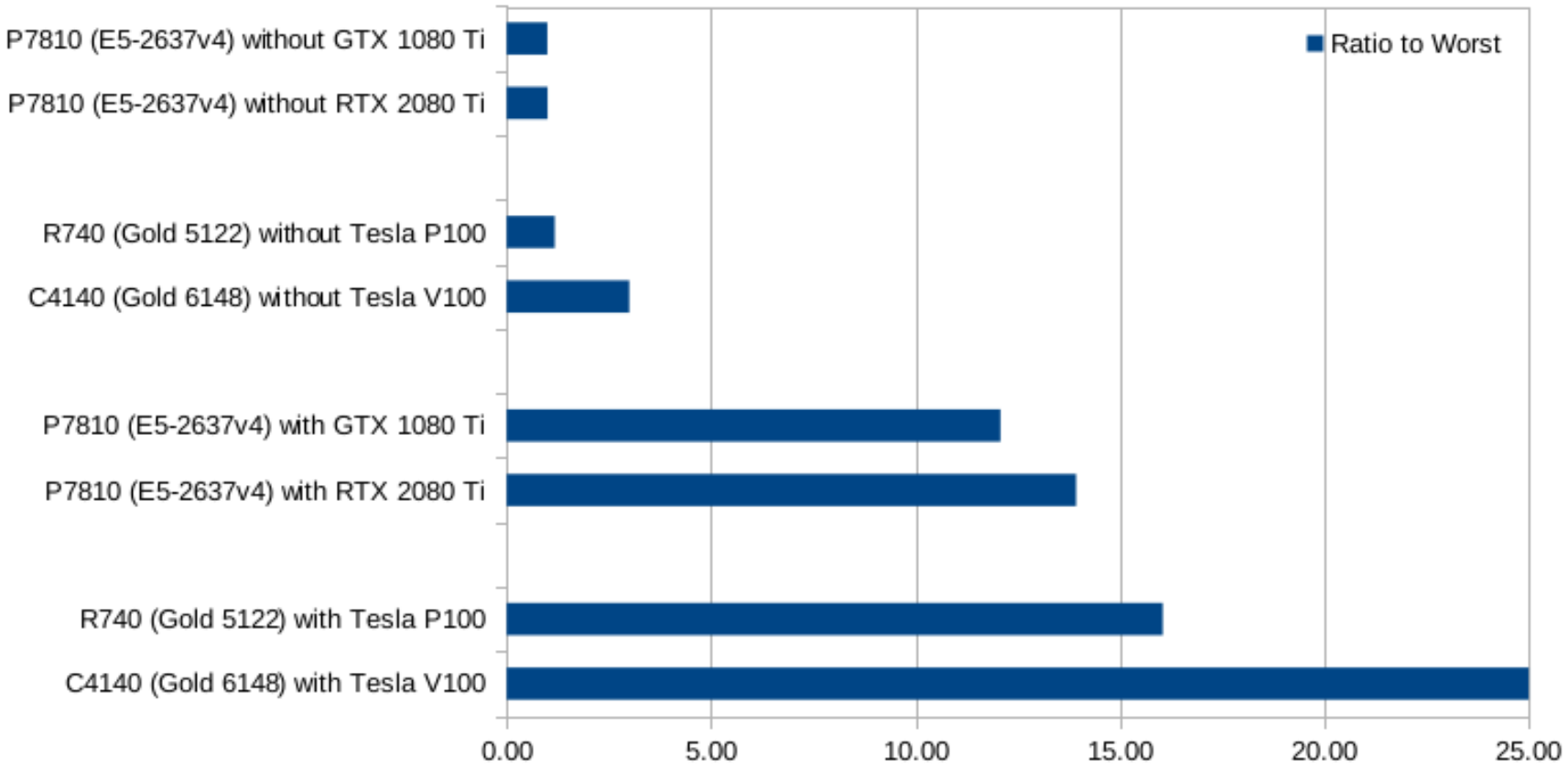
Un « monstre » : une C4140… 2 CPUs 24 cœurs, 4 GPGPU V100

- Sur le code Hoomd (en C++/CUDA, API en Python)

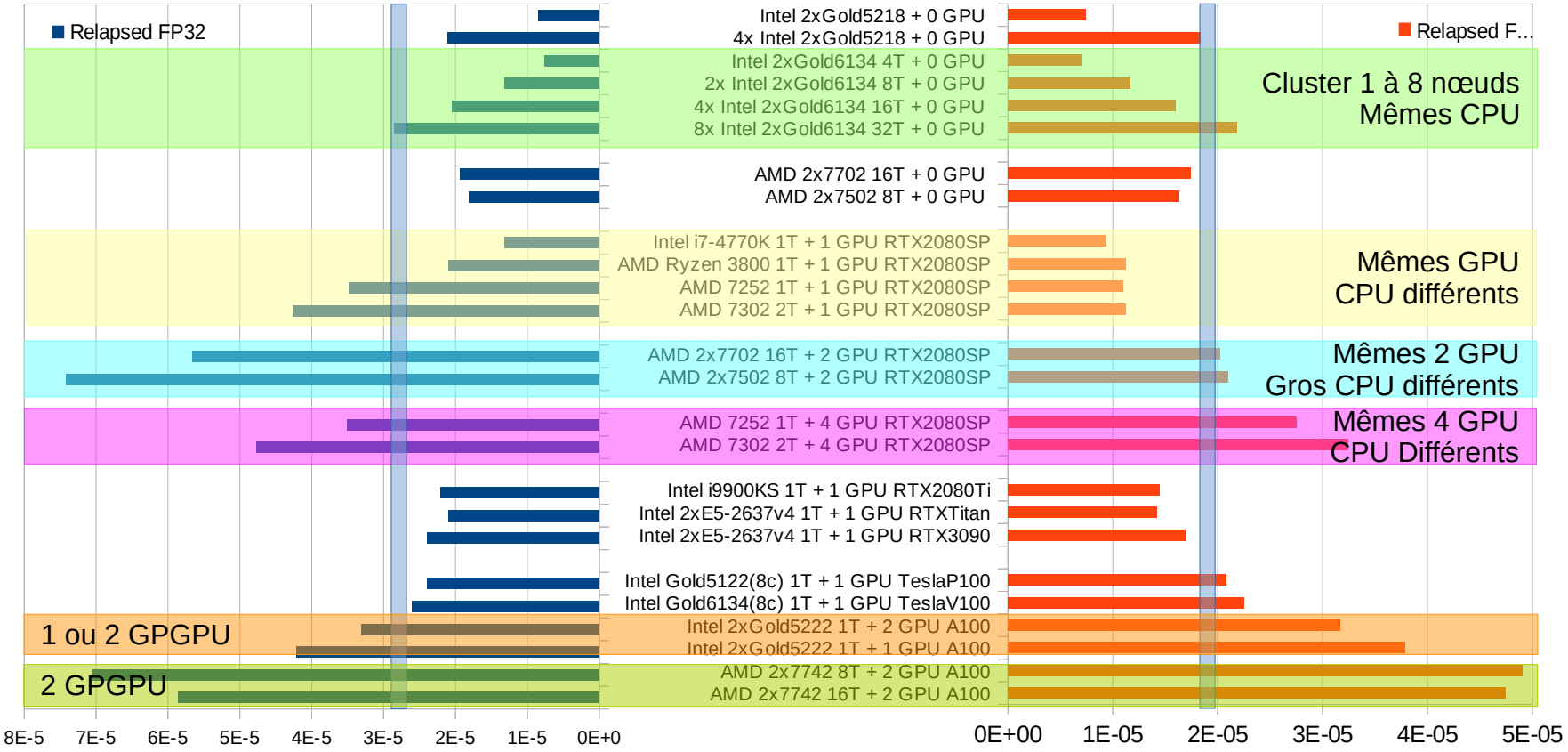


Tensorflow & CIFAR10

- La performance, oui, mais à quel prix ?



Code « métier » trhybride : GENESIS … et un même cas d'usage !

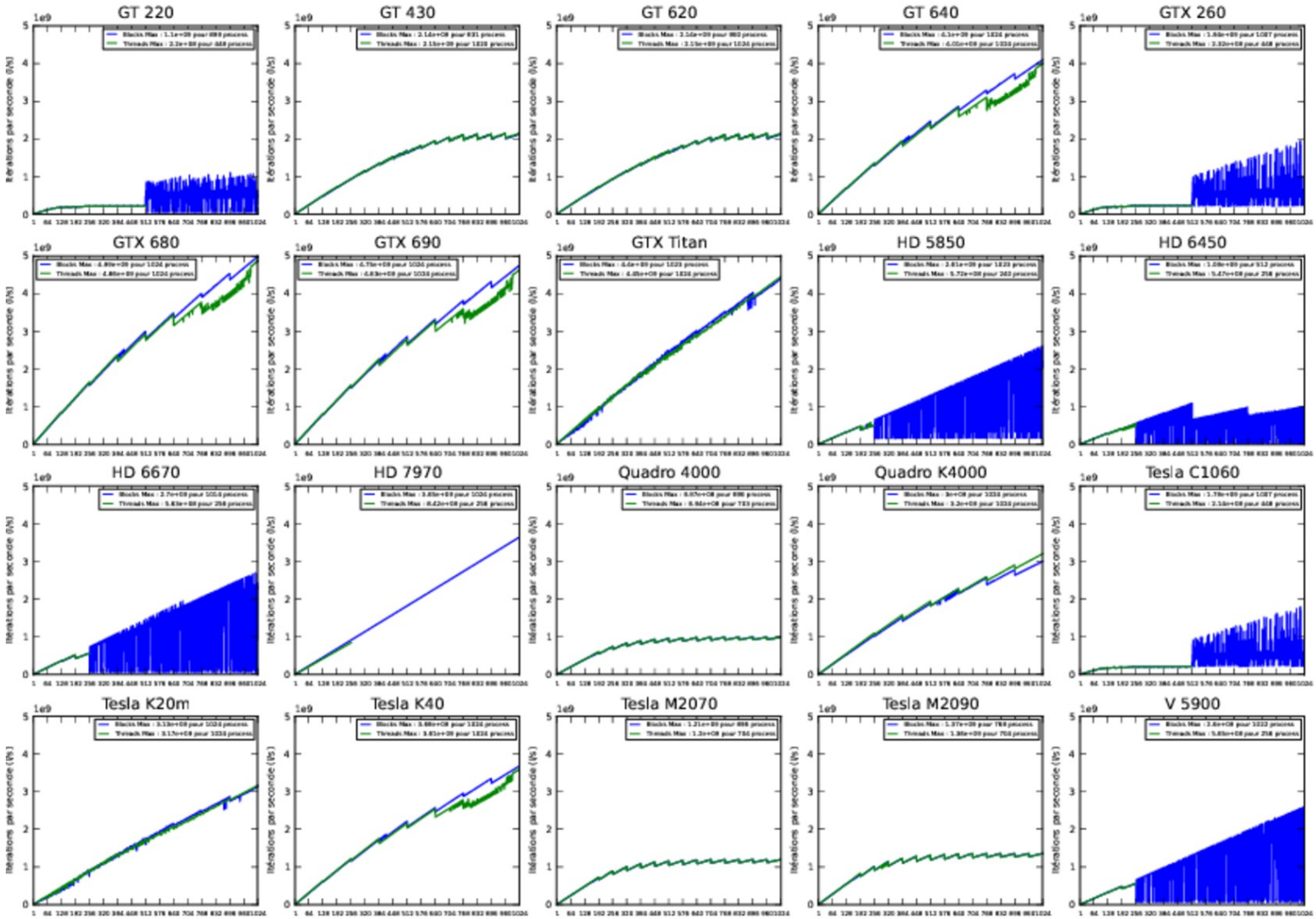


Comme quoi, difficile de se projeter sur une performance...

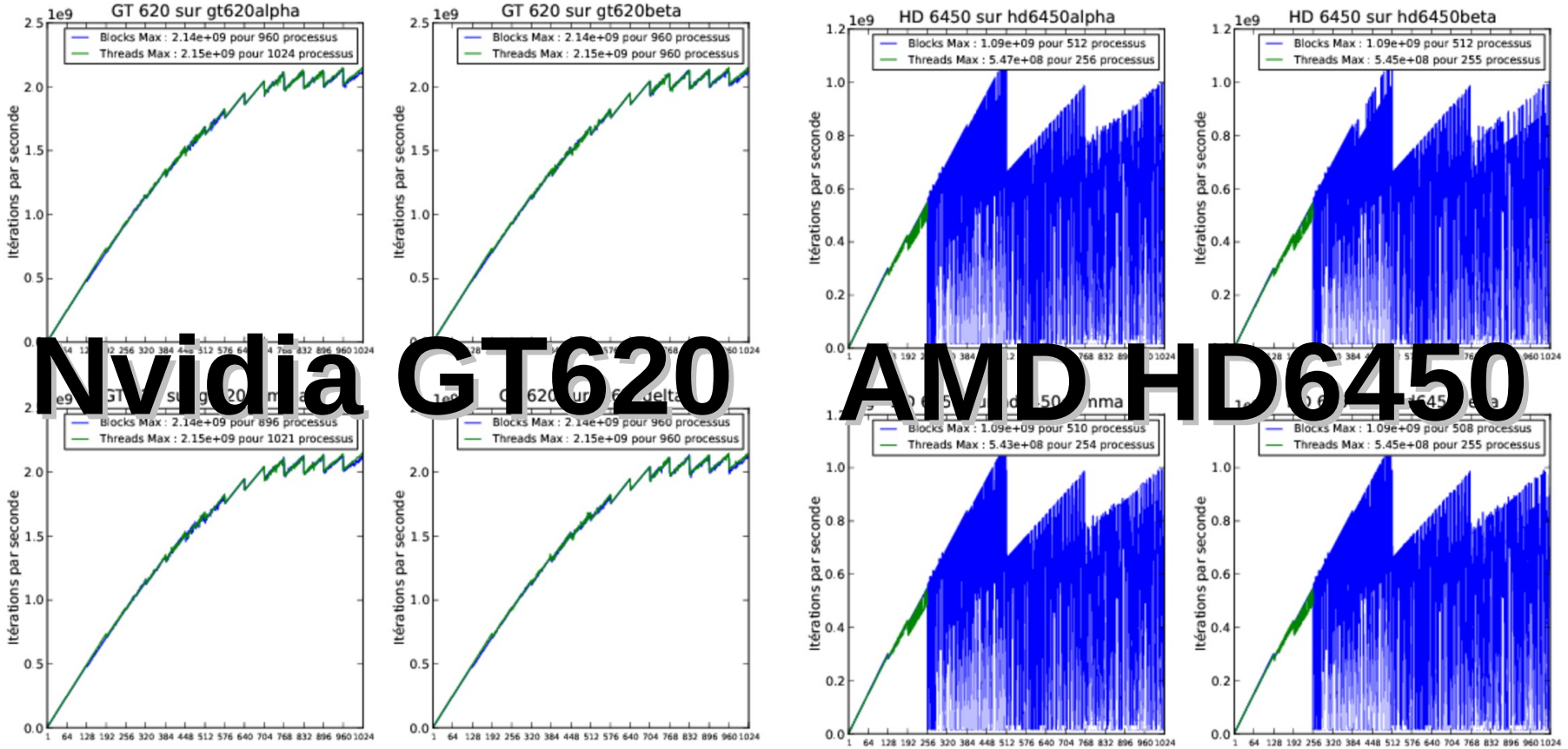
Des « trucs » bizarres… Petit florilège...

- Explorer l’architecture par « reverse engineering »...

Comportement de 20 (GP)GPU vs PR

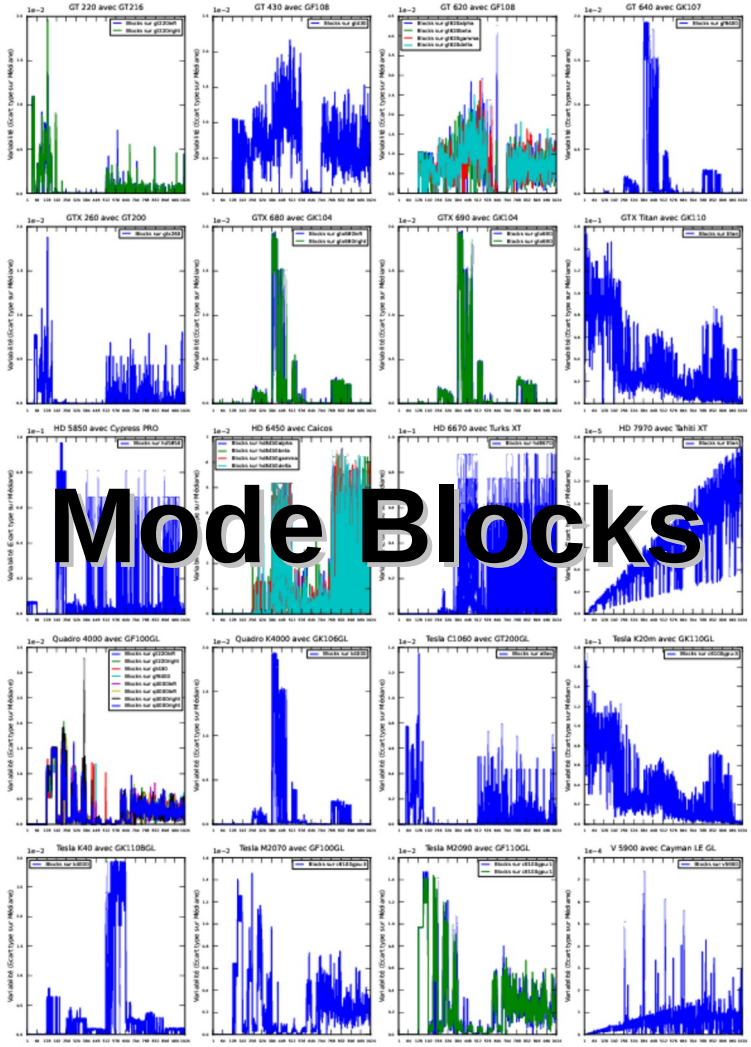


Enveloppes de parallélisme reproductibles ? Pour deux cartes spécifiques...

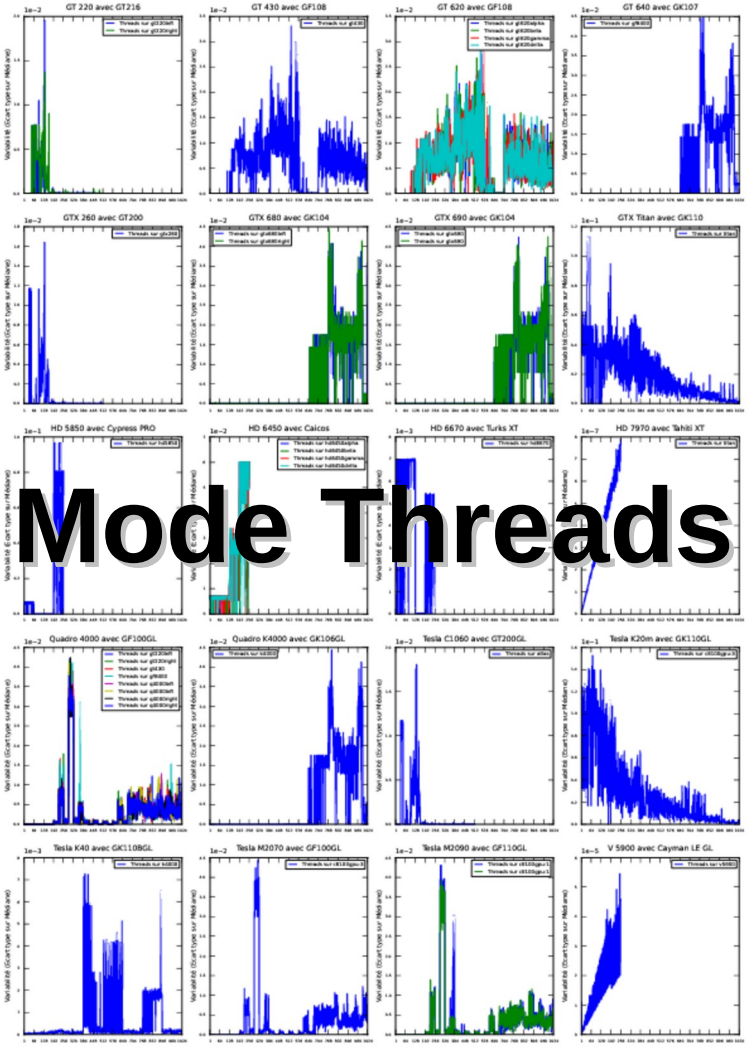


Nvidia GT620 **AMD HD6450**

Variabilité : un critère distinctif ! Quelle étrange propriété :-/ ...



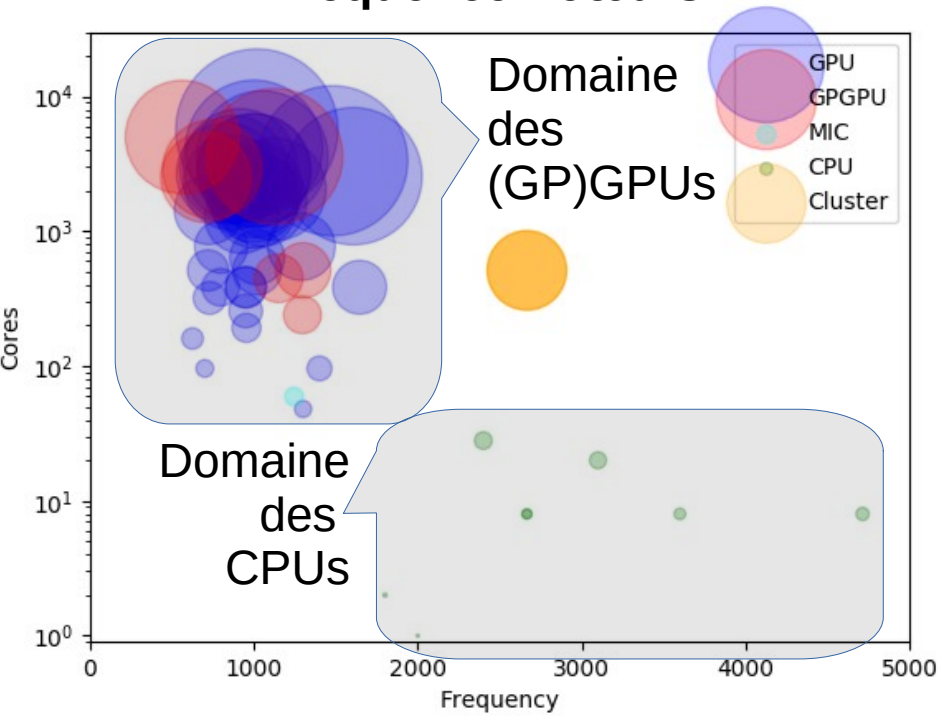
Mode Blocks



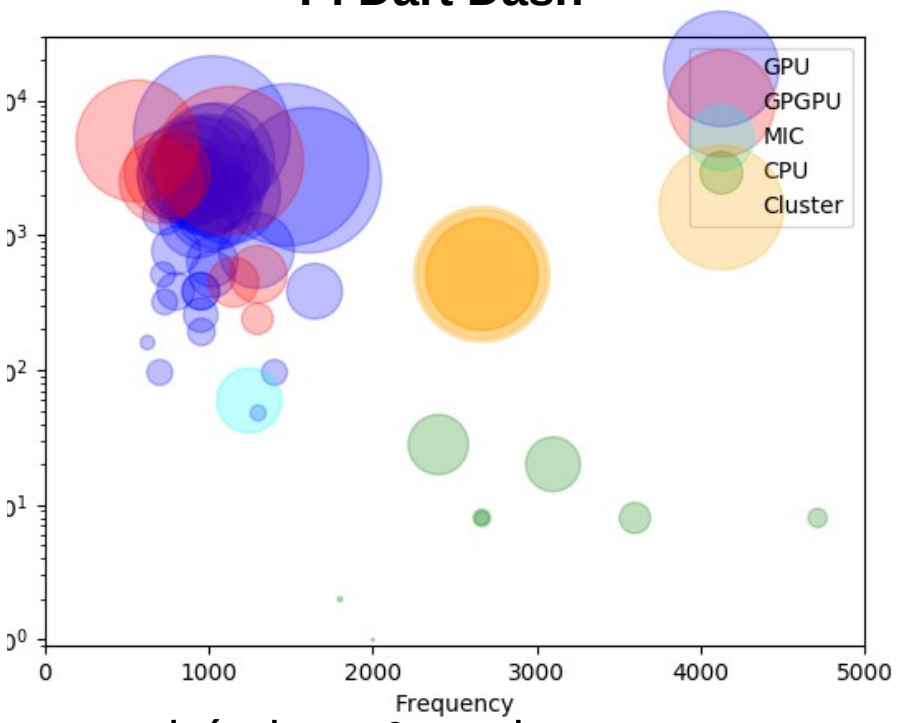
Mode Threads

Représenter les performances... Question de battement, de cœurs ?

Performance Théorique
Fréquence * cœurs



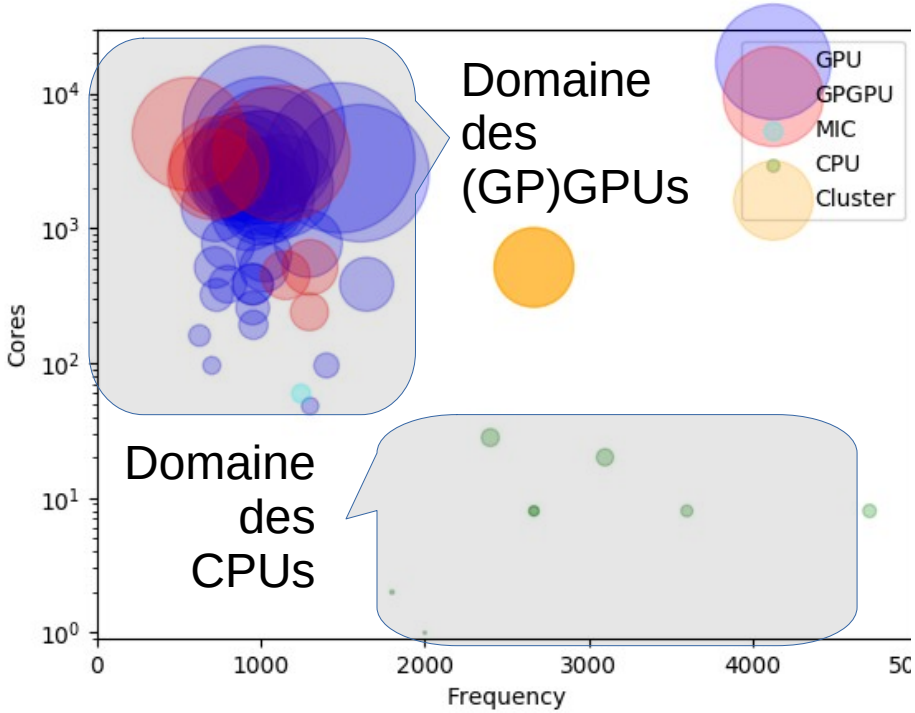
Performance en 32 bits
“Pi Dart Dash”

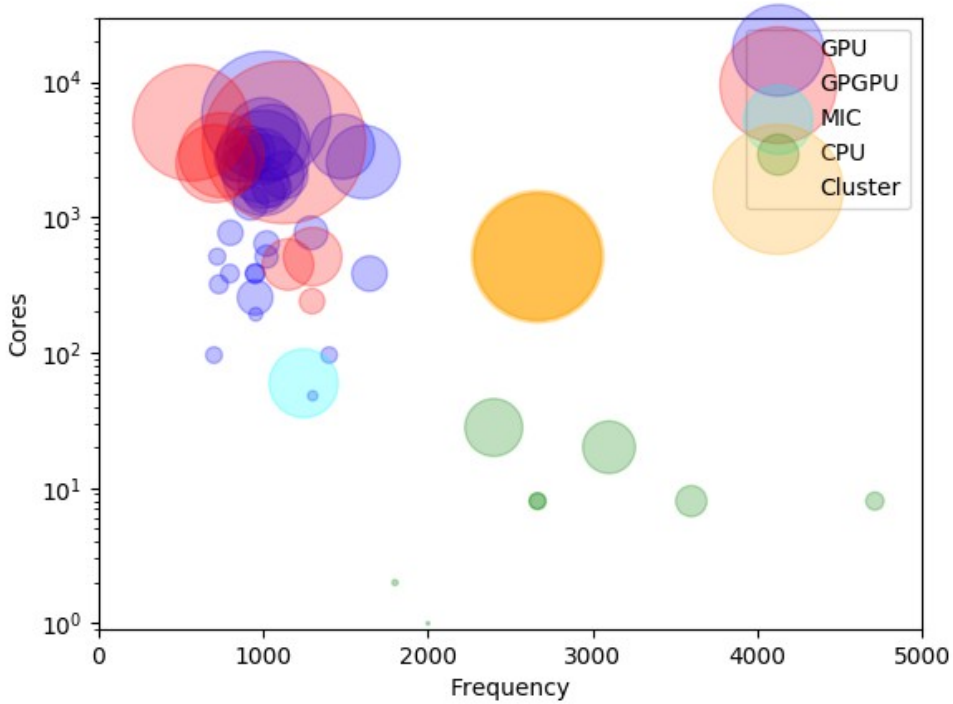


- Sur un GPU, cohérence entre performances théorique & pratique
- Sur un CPU, performance relative meilleure

La performance en informatique

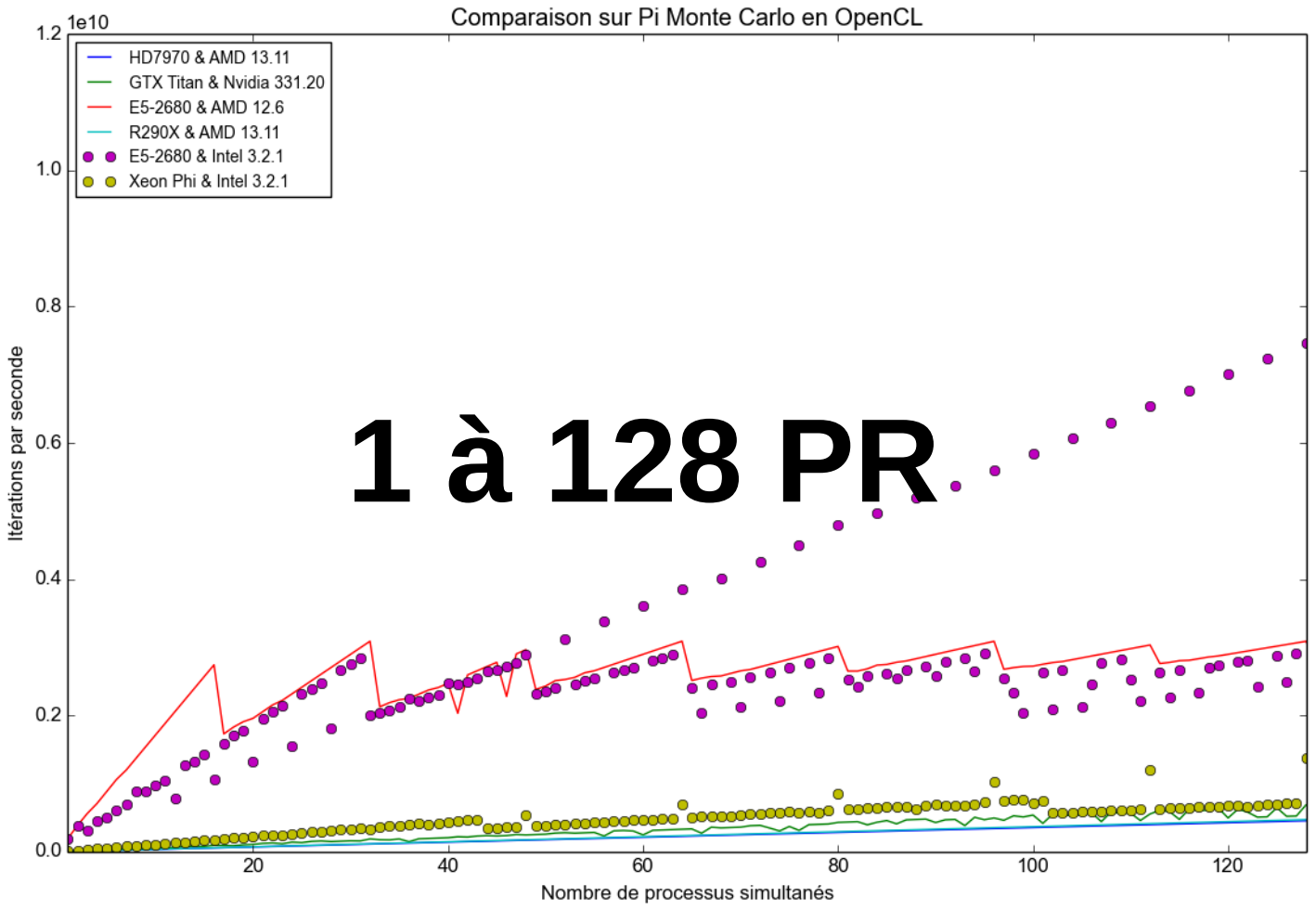
Question de battement, de cœurs, et de bits !



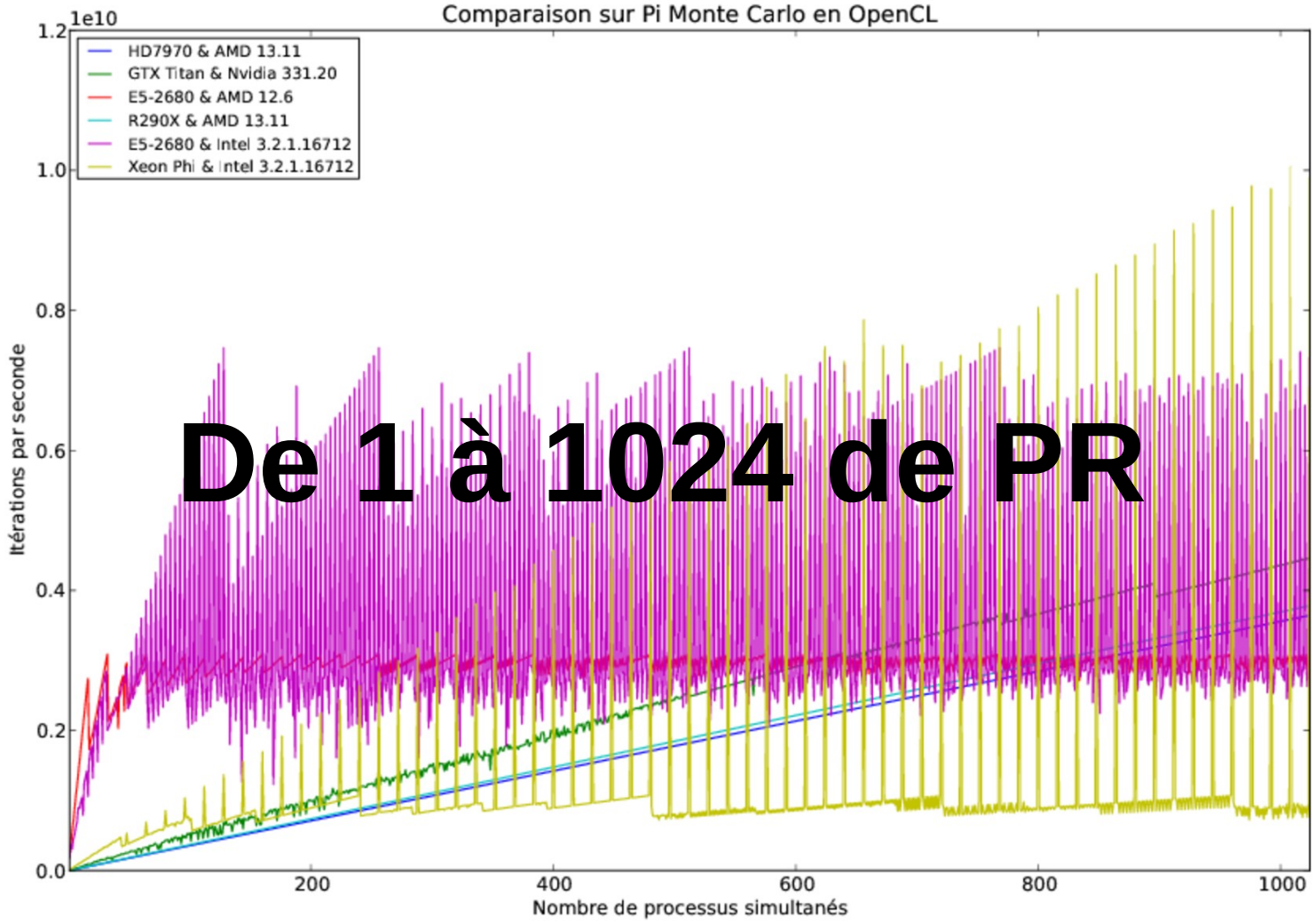


- Sur un GPU, baisse très sensible des performances pour les GPU
- Sur un CPU, performance relative encore meilleure

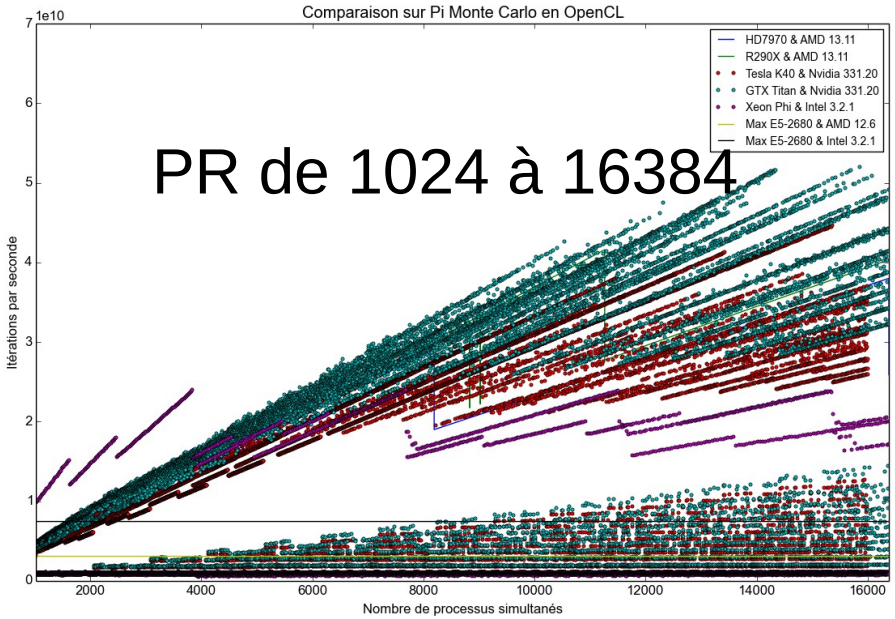
Petite comparaison entre *PU
Bas pararégime : le règne du CPU

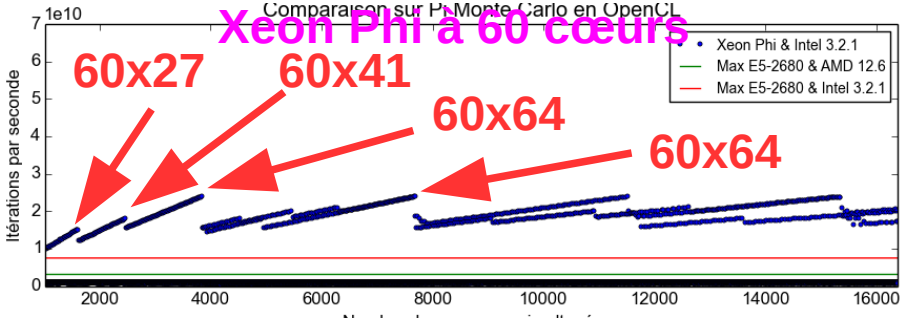


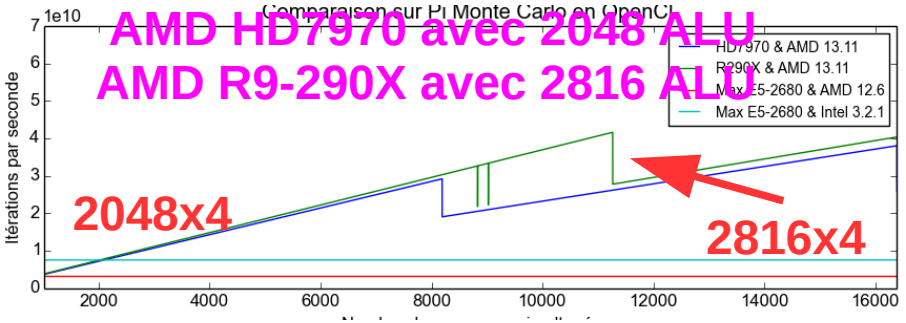
Les processeurs toujours efficaces
Le Xeon Phi « sort du bois »

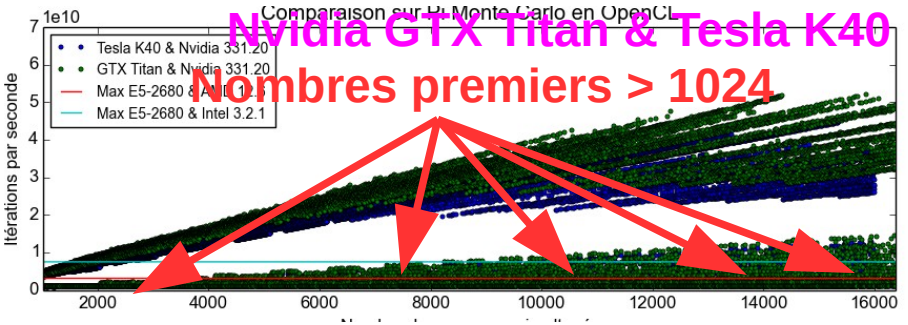


Exploration profonde de PR ParaRégime de 1024 à 16384

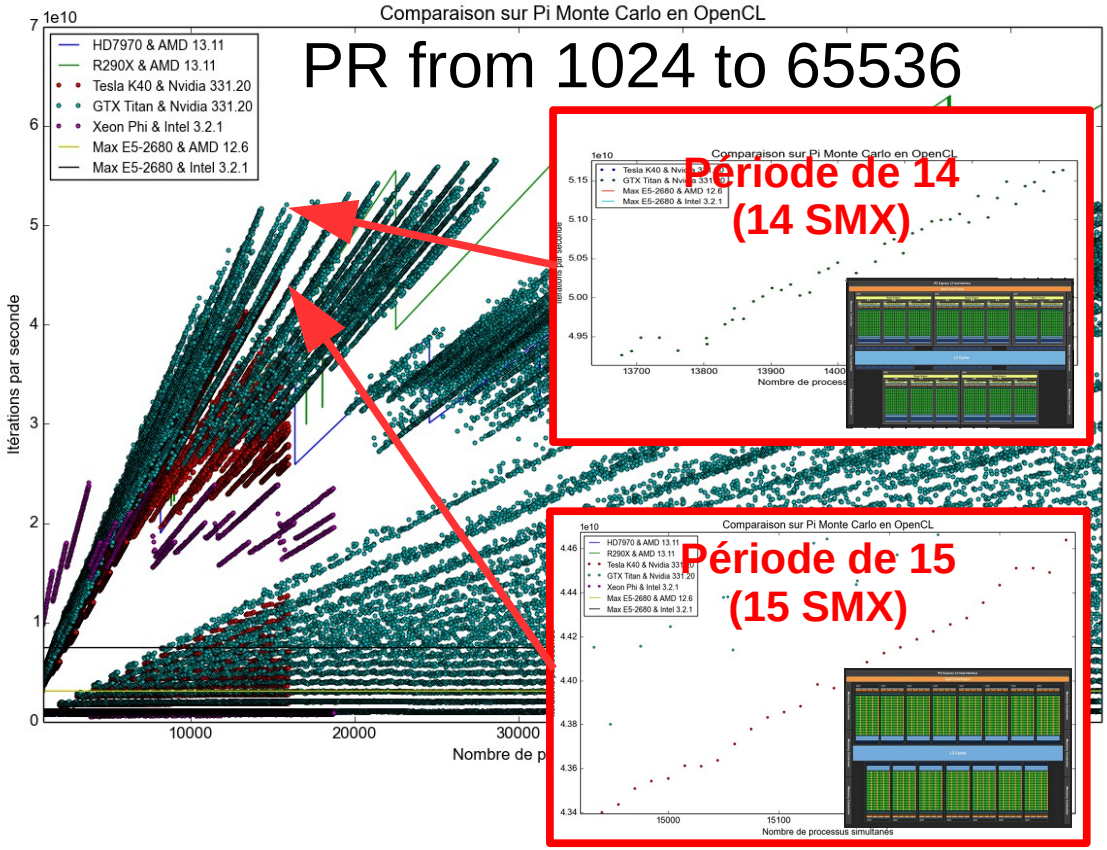




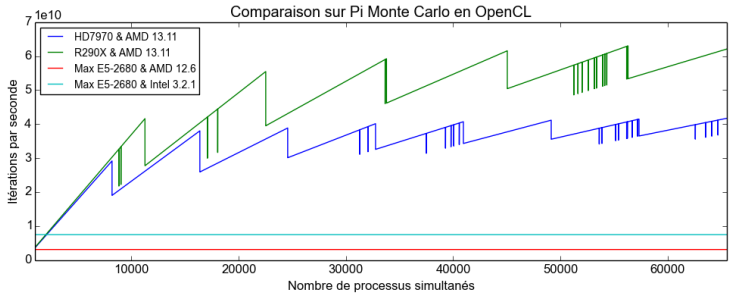




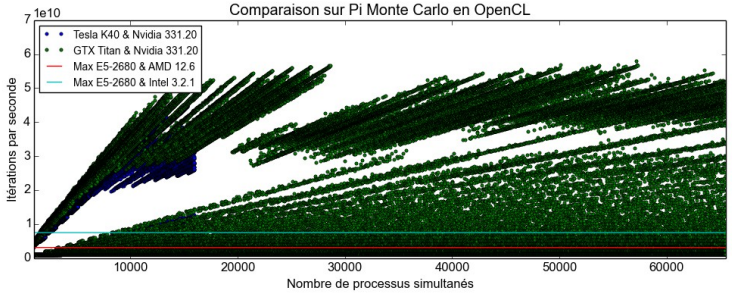
Investiguer la structure interne Par l'exécution de Pi Dart Dash



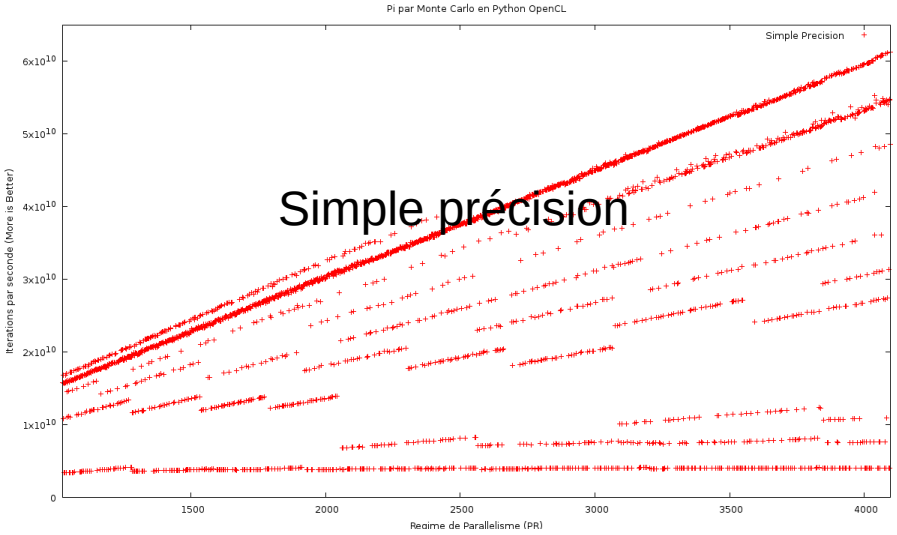
AMD HD7970 & R9-290
Longue Période : 4x nombre d'ALU



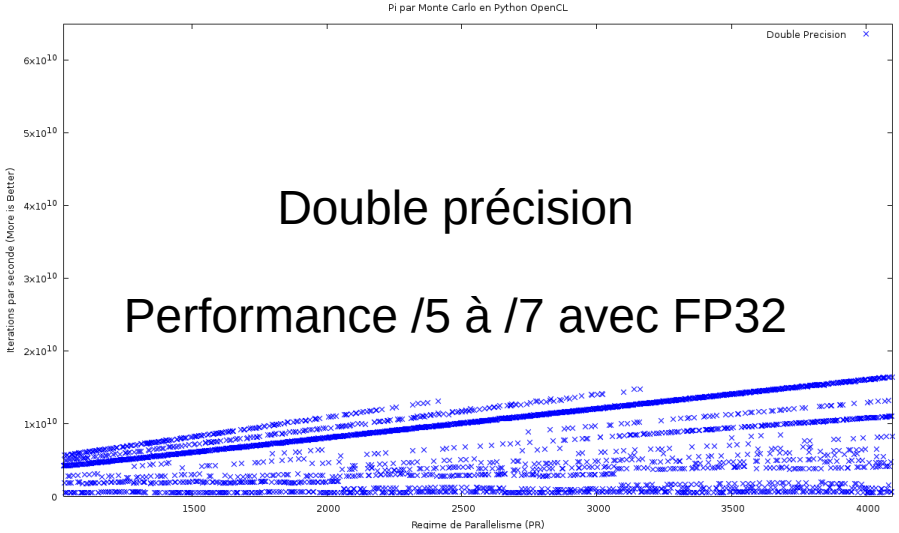
Nvidia GTX Titan & Tesla K40
Courte Période : nombre d'unités SMX



Et les dernières Nvidia GTX1080(Ti)
Toujours affectées de bizarreries ?

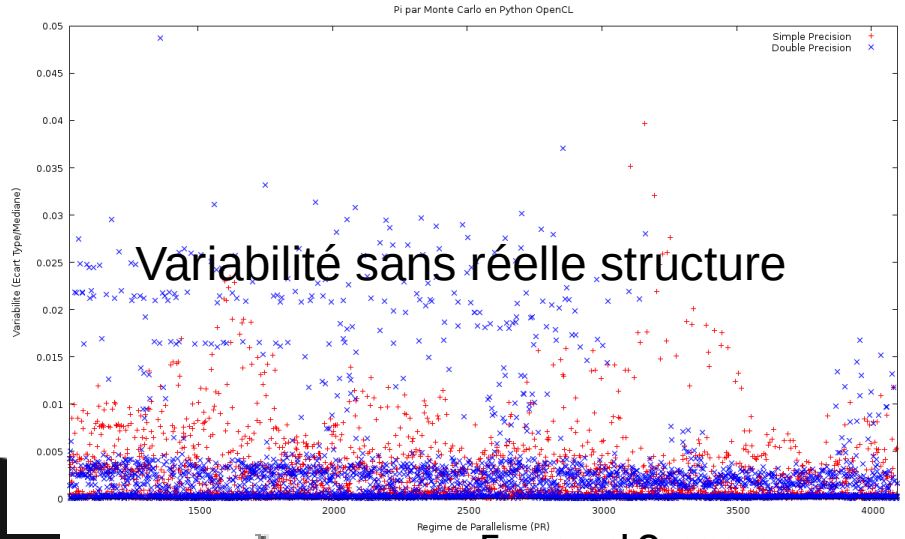


Simple précision

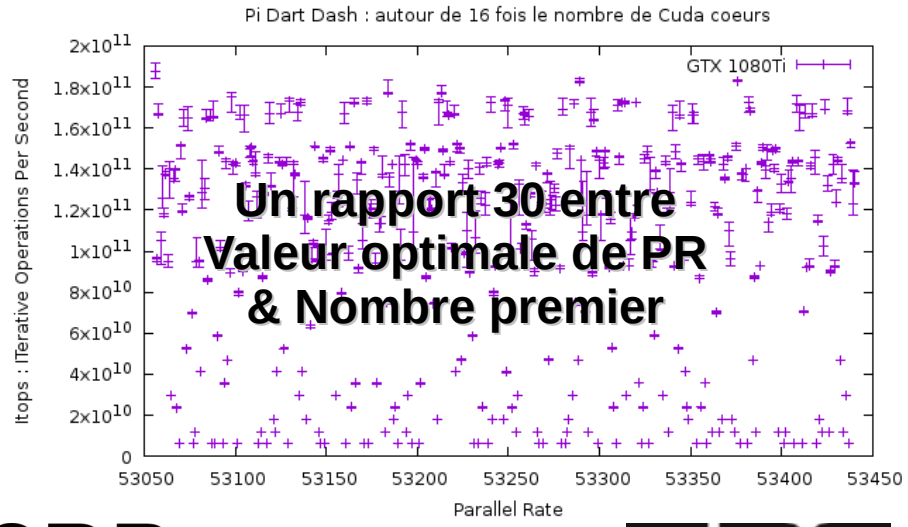


Double précision

Performance /5 à /7 avec FP32

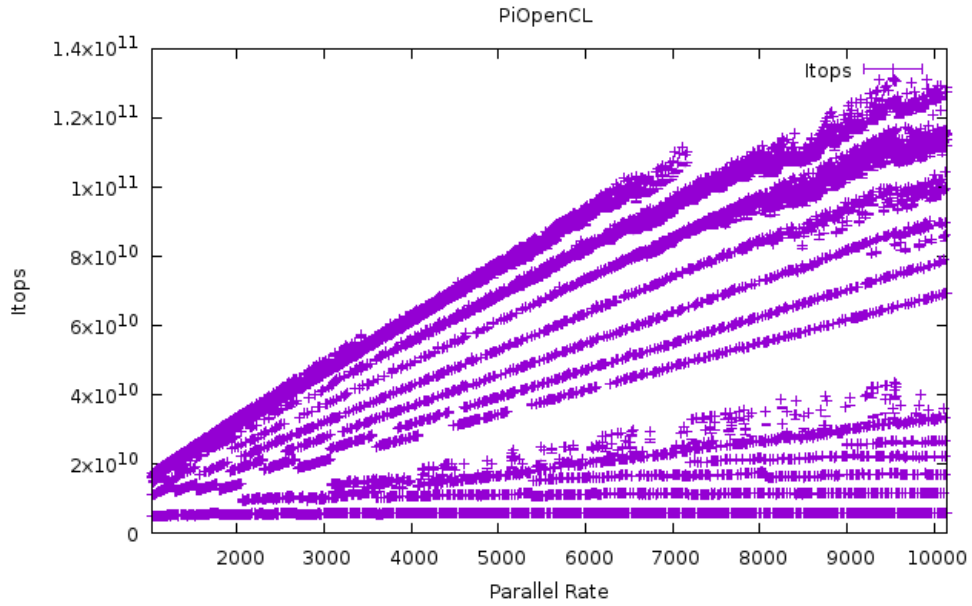


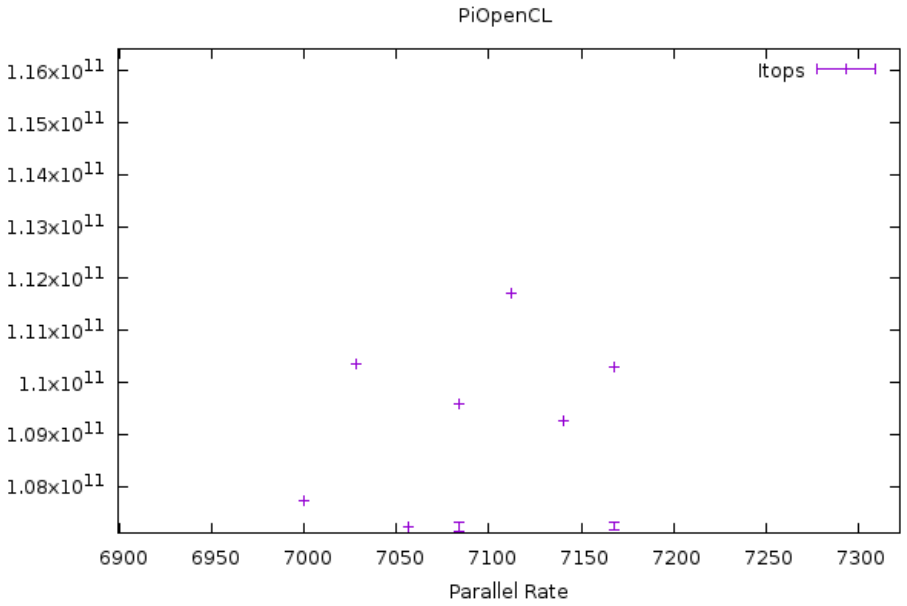
Variabilité sans réelle structure



Un rapport 30 entre Valeur optimale de PR & Nombre premier

Et en passant sur un PiOpenCL en C pur ?

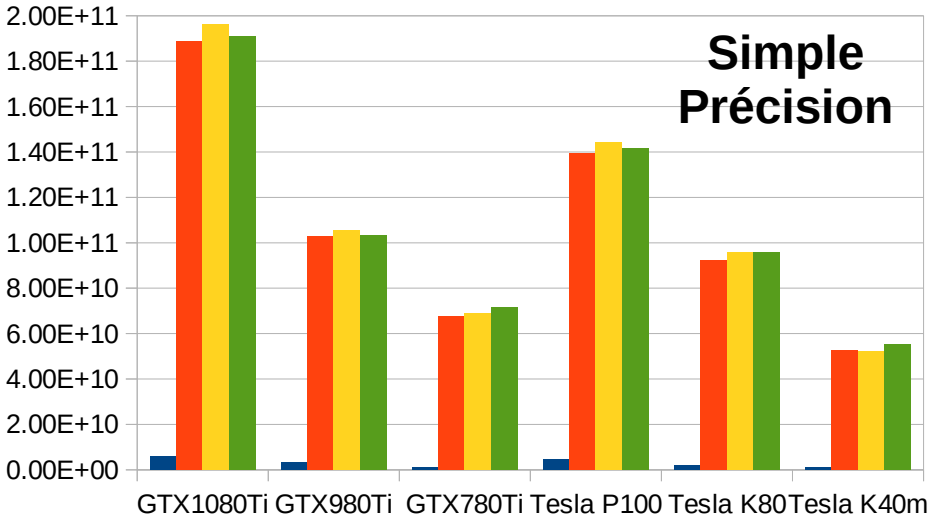


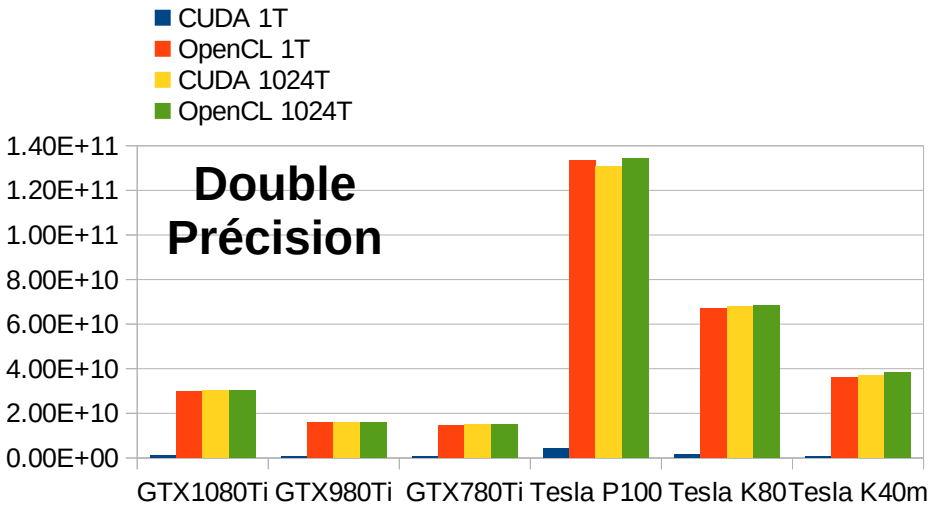


- Les mêmes détections de nombres premiers
- Les mêmes max (x2 le nombre de shaders)

CUDA vs OpenCL : le match

- 6 cartes : 3 GPGPU, 3 GPU, 3 générations
- Un régime de parallélisme mixte :
  - Entre Blocks & Threads ou entre Work Items & Threads
  - (Blocks,Threads)=(CudaCores,1024) ou (CudaCores*1024,1)
  - CudaCores : 2816, 2880, 3584, 2x2496



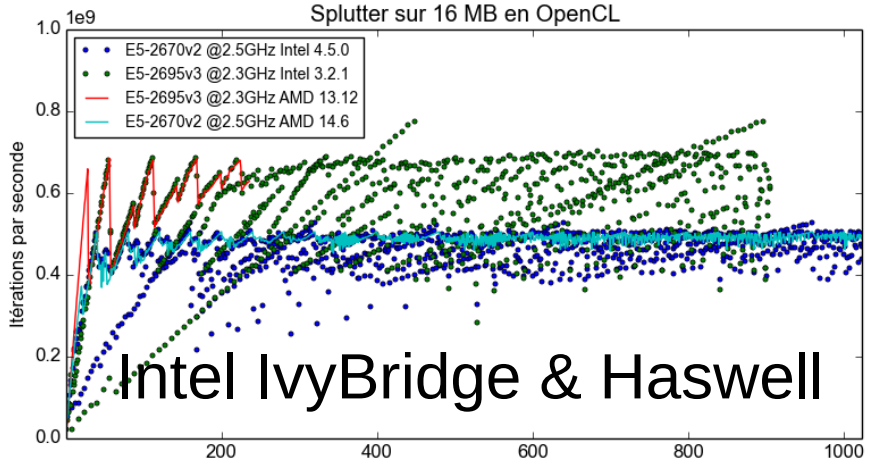


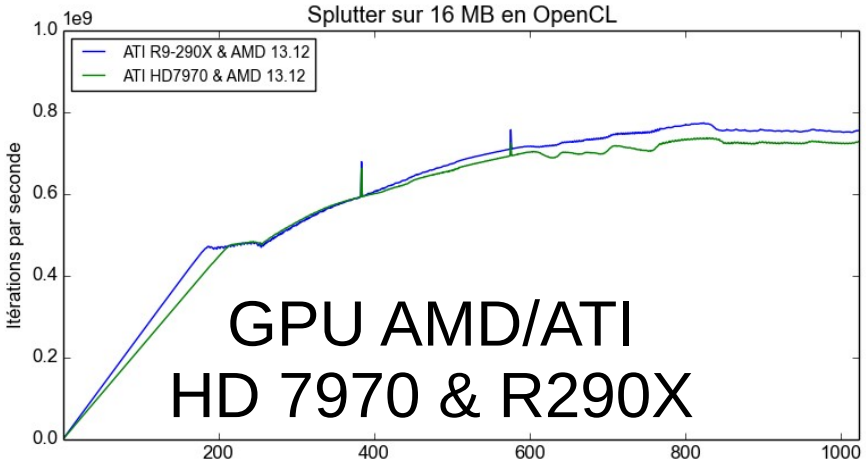
Bref, pour ce code, CUDA~OpenCL, même mieux !

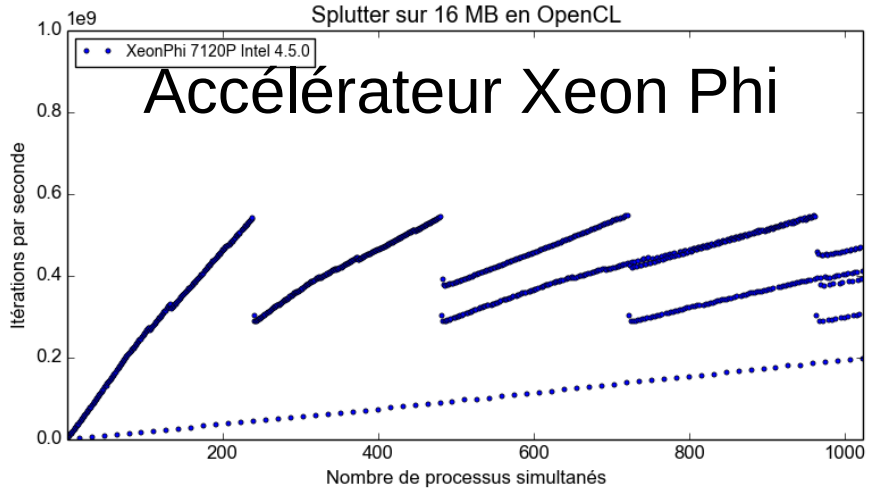
Un test « Memory Bound » Le « postillonneur »...

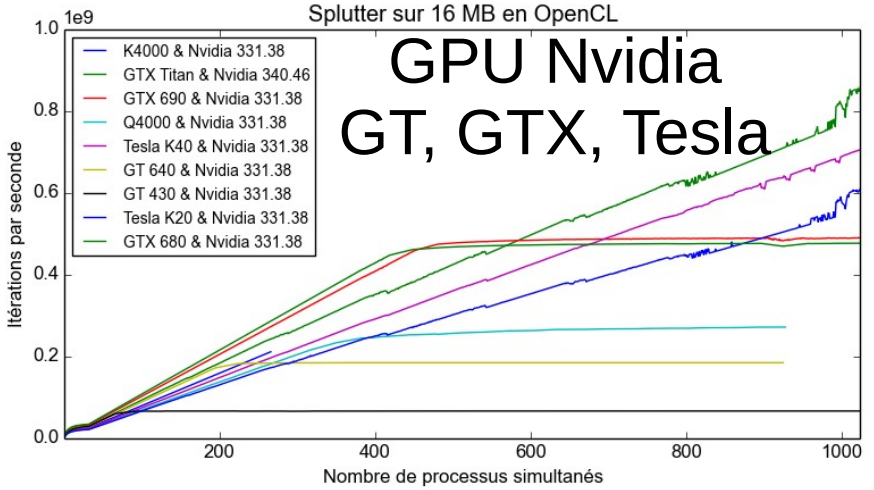
- Le programme :
  - Je prends un espace mémoire
  - Je le transfère sur le périphérique (GPU ou CPU) : H2D
  - Pour chaque tâche parallèle, nombre équivalent d'itérations
    - Je tire un nombre aléatoire modulo l'espace mémoire
    - J'incrémente avec atomic_inc l'espace mémoire
  - Je recupère l'espace mémoire du périphérique : D2H
- La vérification : je somme les éléments de l'espace
  - Je dois retomber sur le nombre d'itérations sélectionnés
- Les observables : les 3 temps…
  - Le temps H2D, le temps de postillonnage, le temps D2H

Le « postillonneur » en OpenCL Il y a 4 ans...









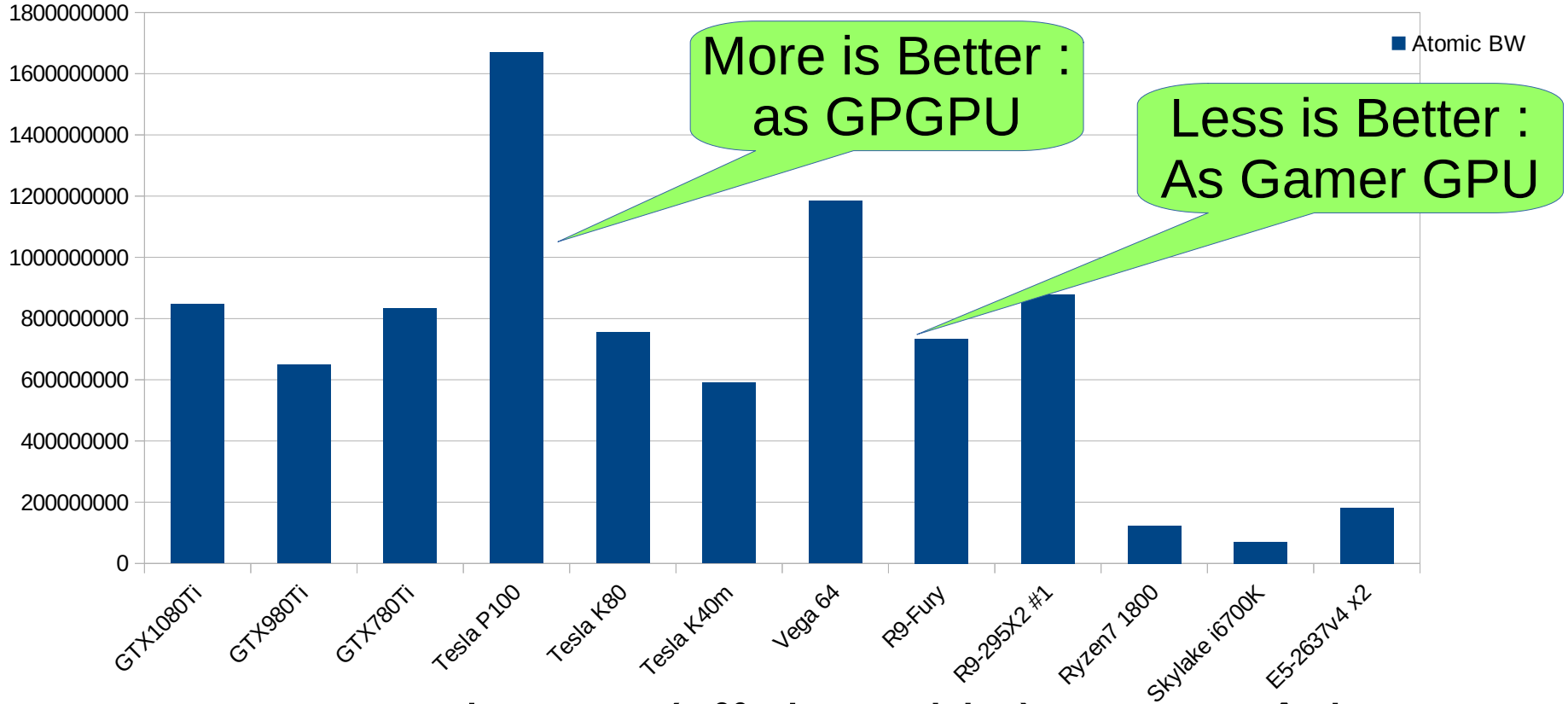
Et pour les GPU & CPU utilisés ? D'abord pour les 9 GPU...

- Un espace de 2GB (2^{29} de 32 bits), 32768 tâches
- 8.5 Millards d'itérations (16 par tâche)

Et pour les GPU & CPU utilisés ? Les 9 GPU + les 3 meilleurs CPU

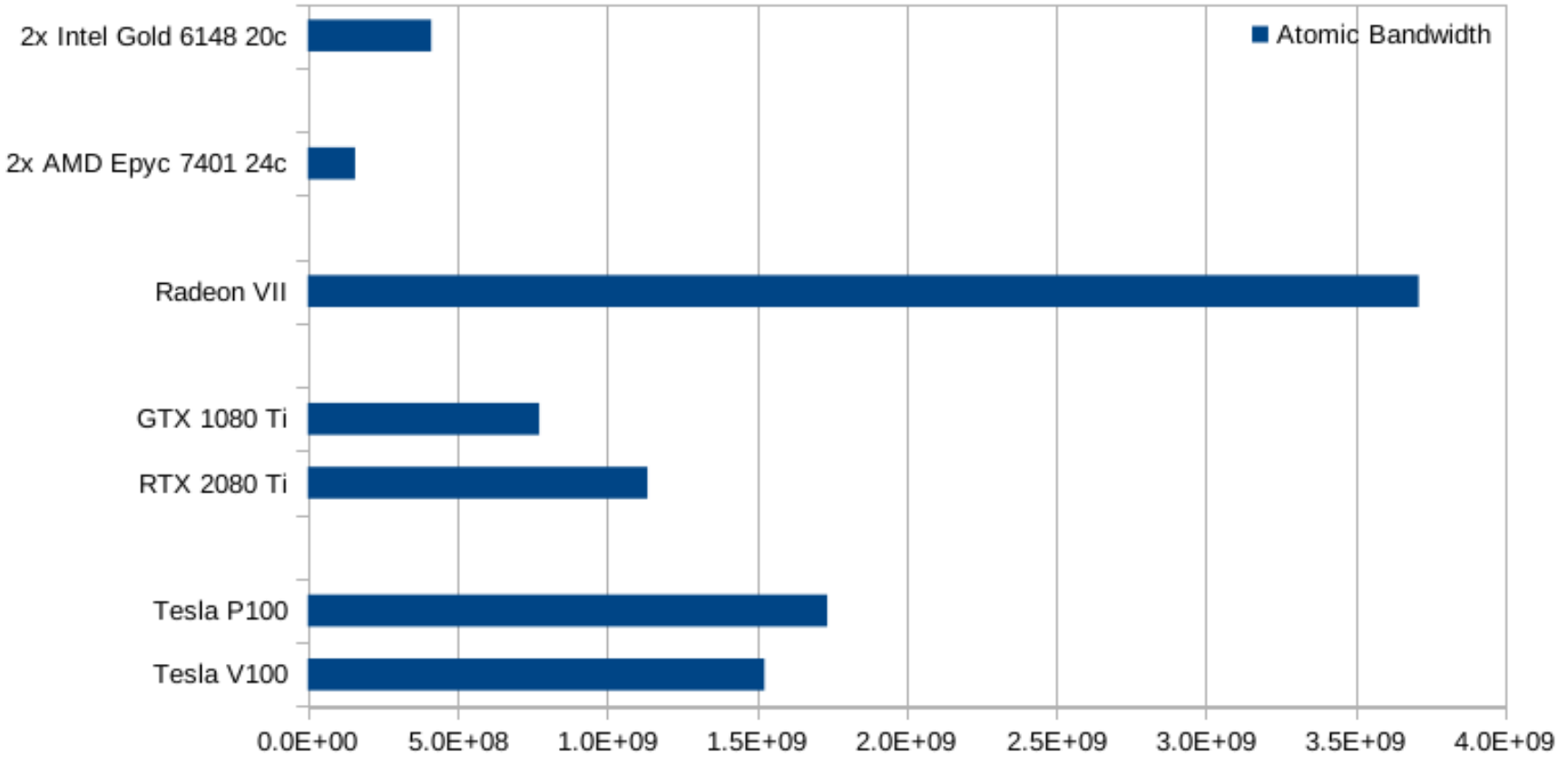
- Un espace de 2GB (2^{29} de 32 bits), 32768 tâches
- 8.5 Millards d'itérations (16 par tâche)

Et pour les GPU & CPU utilisés ? Les 9 GPU + les 3 meilleurs CPU



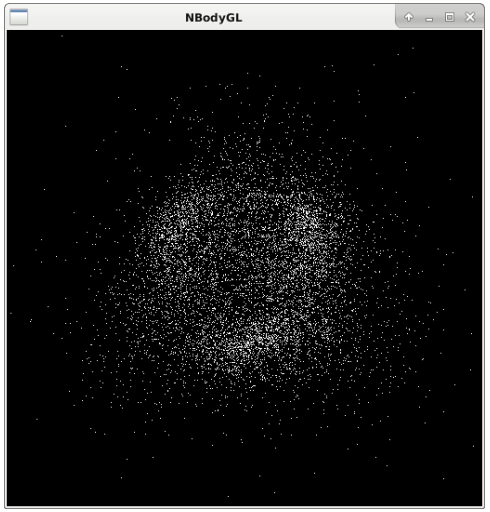
- Un espace de 2GB (2^{29} de 32 bits), 32768 tâches
- 8.5 Millards d'itérations (16 par tâche)

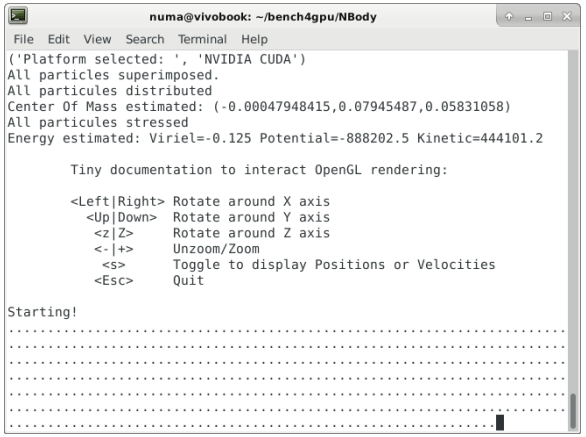
Sur des architectures « récentes » Comme quoi, avec une AMD...



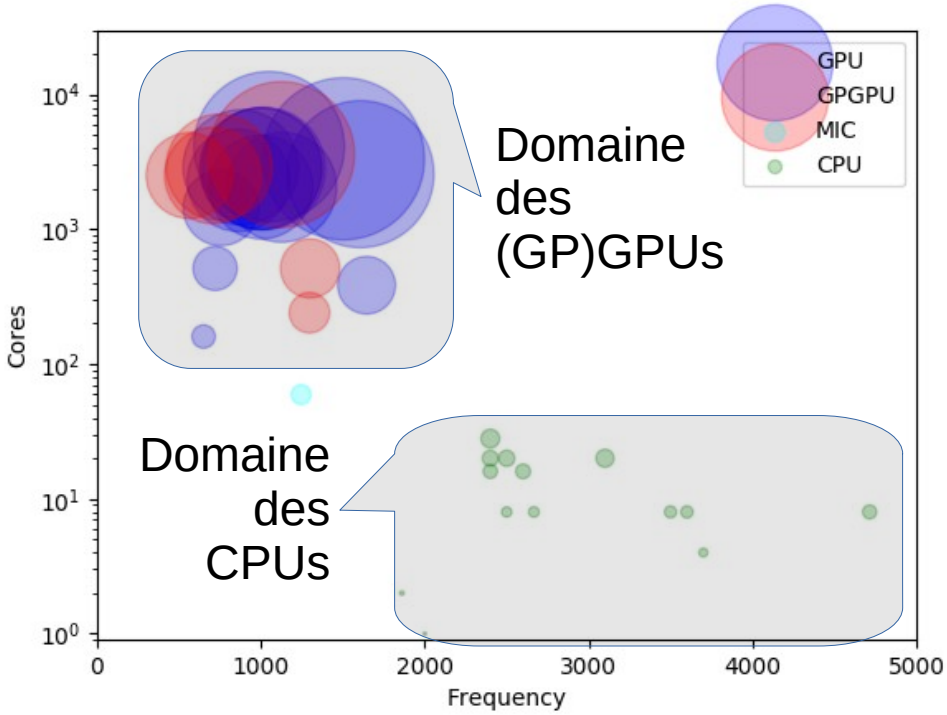
Retour à la physique
Un code newtonien N-corps...

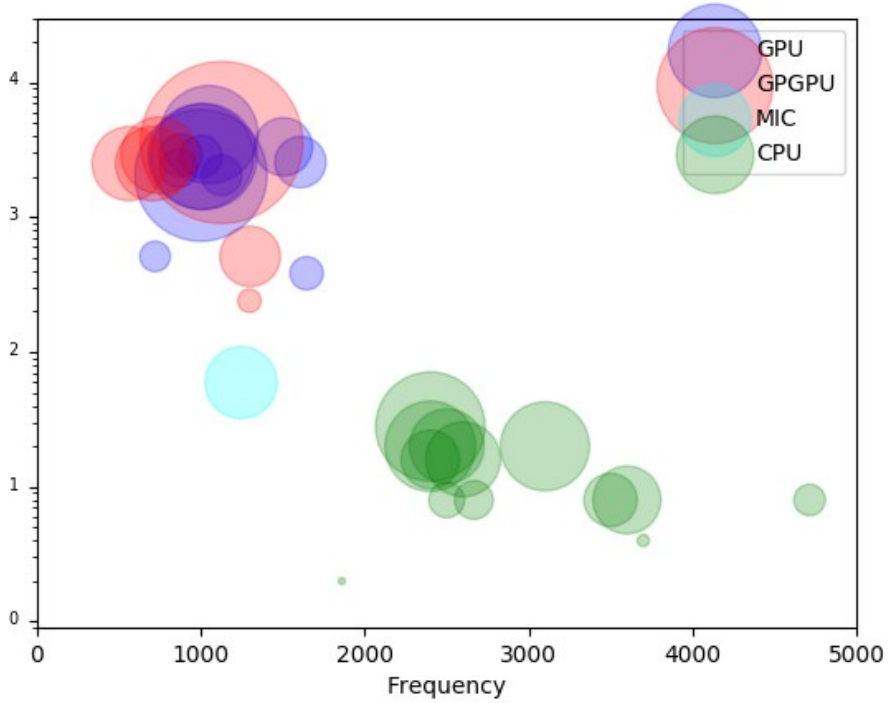
- Passage sur un code « grain fin »
- Code N-body très simplement implémenté
  - Seconde loi de Newton, système autogravitant
- Méthodes d'intégration différentielle :
  - Euler Implicite, Euler Explicite, Heun, Runge Kutta
- Données d'entrées :
  - Physiques : nombre de particules
  - Numerique : pas d'intégration, nombre de pas
  - Architecture : influence de la nature du flottant FP32, FP64





La performance en informatique
Pour un code plus physique...

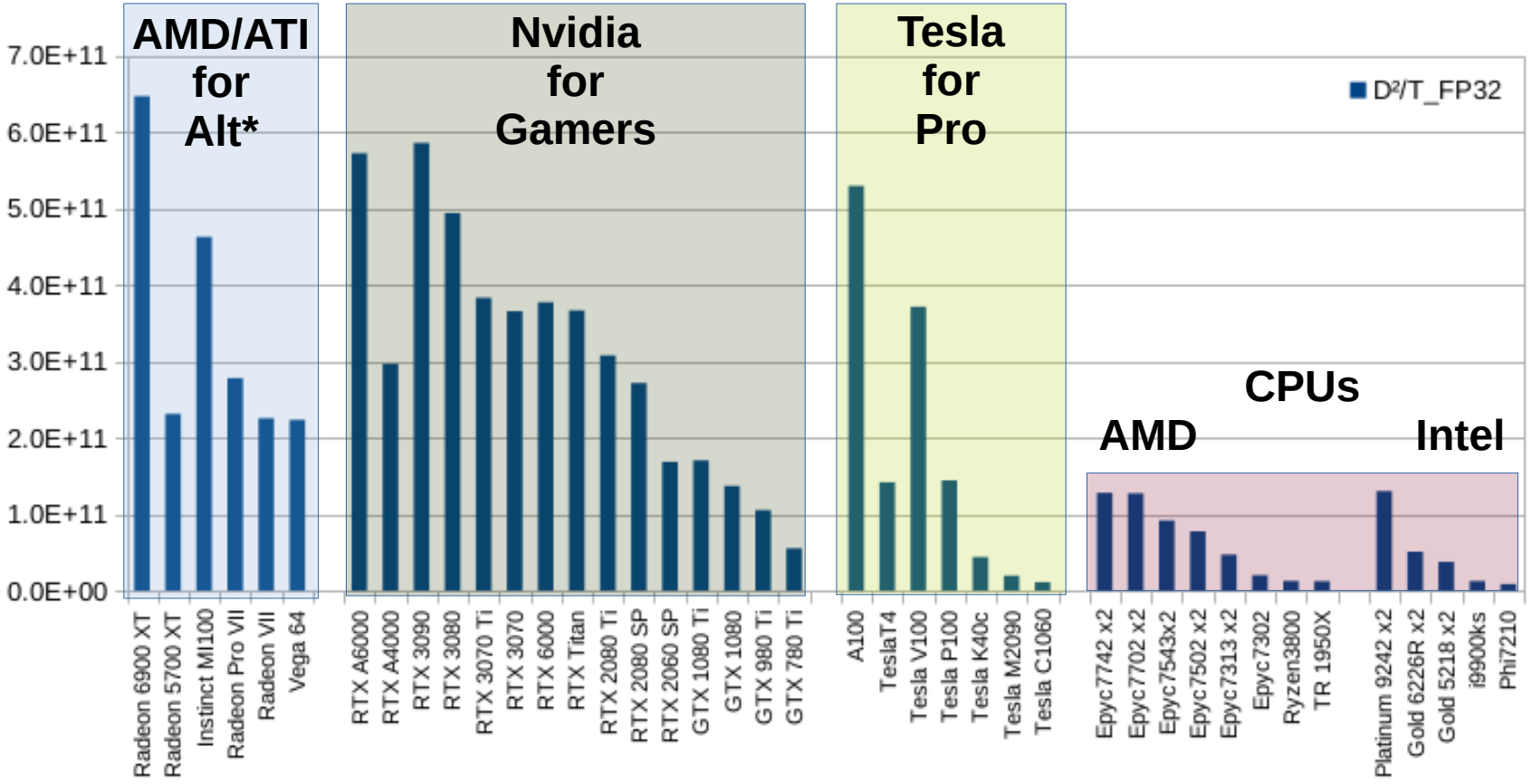




- Sur un GPGPU, performance stable
- Sur un GPU, baisse drastique de la performance (division par 20 en DP)
- Sur un CPU, performance relative meilleure

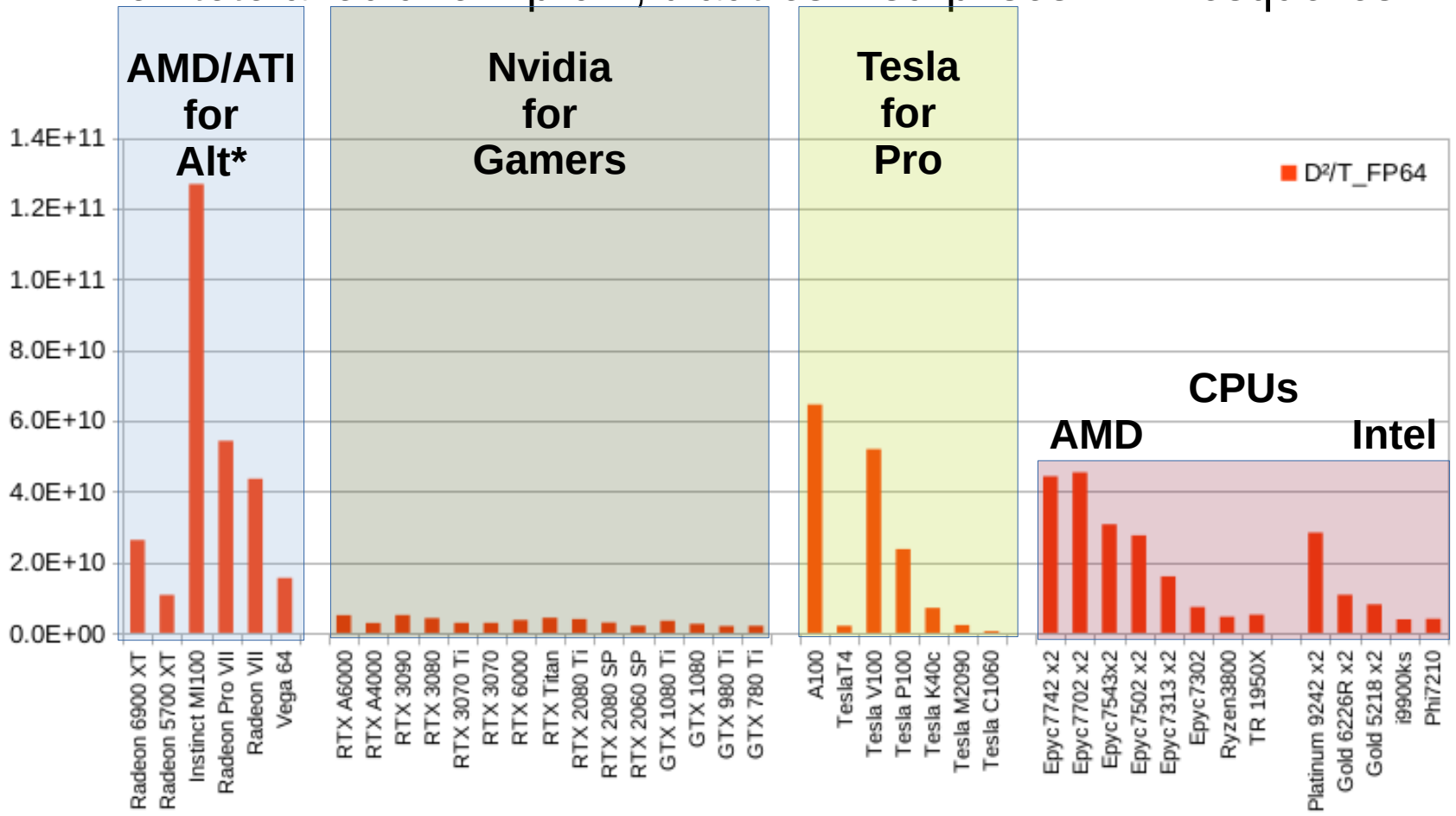
Nbody en FP32
de 32 à 1048576 particules

- AMD en tête avec une « gamer », des « surprises »… Lesquelles ?



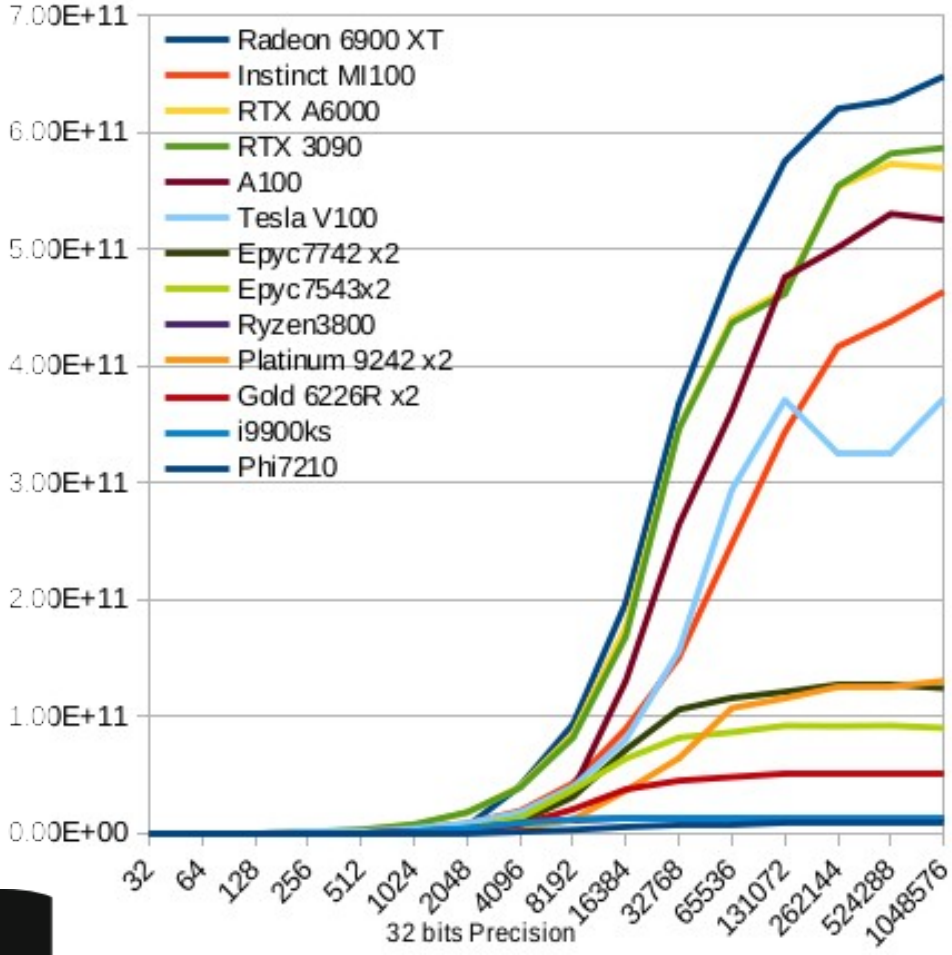
Nbody en FP32
de 32 à 1048576 particules

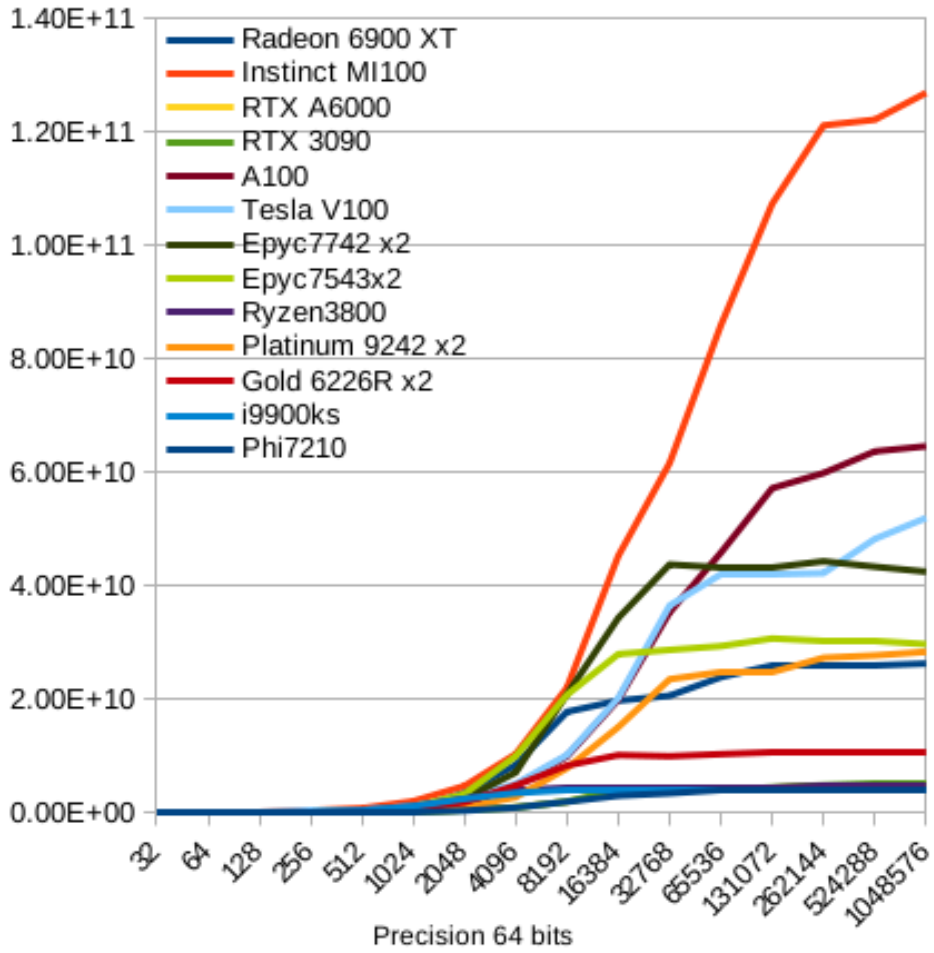
- AMD en tête avec une « pro », d'autres « surprises »… Lesquelles ?



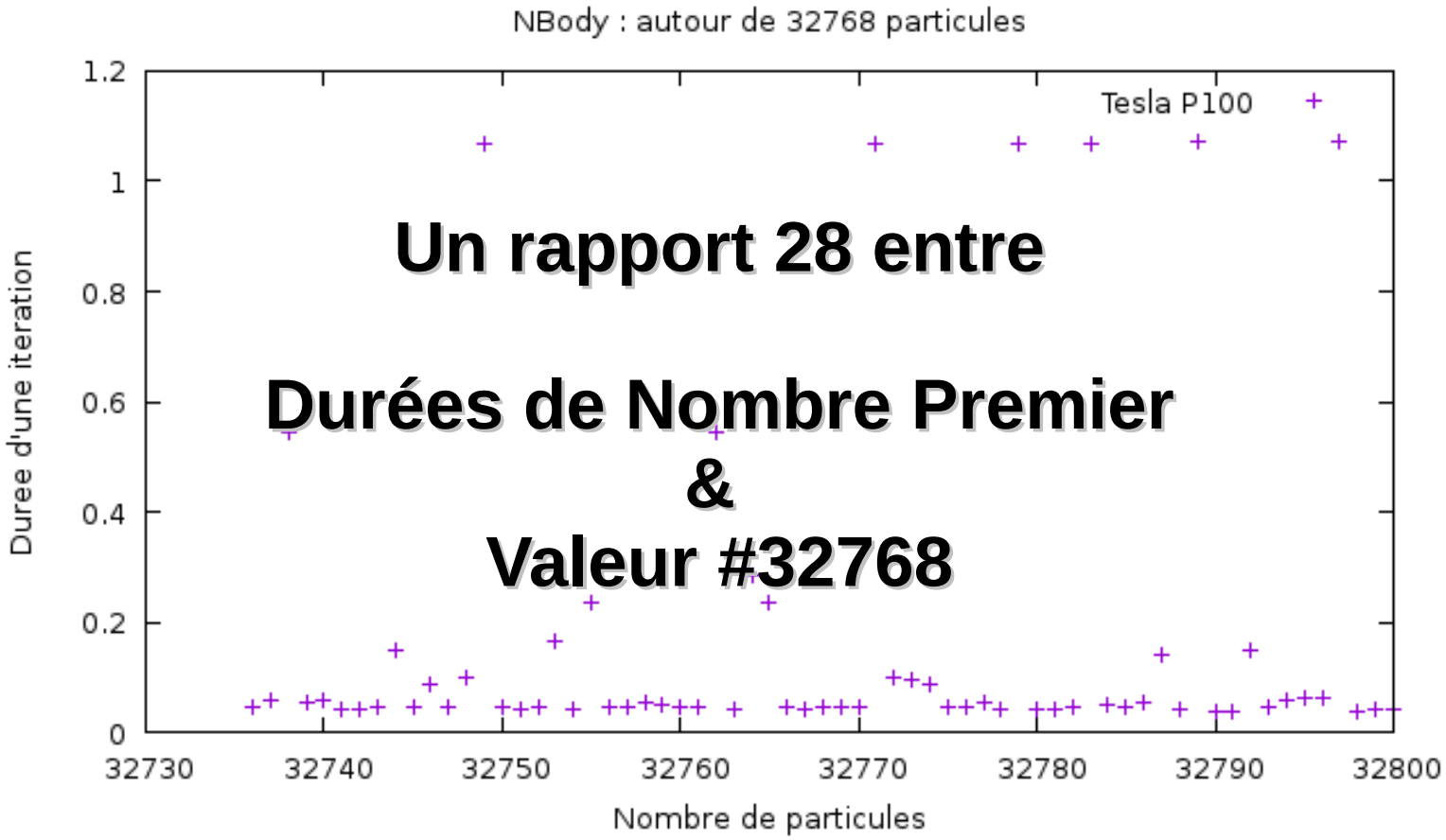
Des progressions en performance On retrouve les > 200000 tâches !

- Pour les CPU : >32768. Pour les GPU : >262144 !





Et les effets « prime numbers » pour les cartes Nvidia ?



Quelques maximes...

- A chaque usage, son architecture optimale !
  - Un usage, c'est l'association d'une recette et ses ingrédients
- A chaque GPU, des régimes de parallélisme optimaux !
  - Encore faut-il les rechercher…
- A chaque tâche élémentaire, une charge conséquente !
  - Plusieurs centaines d'opérations minimum...

Mais à quoi ça sert tout ça ? Caractériser & éviter les regrets...

- Ce qu'il faut retenir :
  - Dans une machine, en 2022, la puissance « brute » est dans le (gros) GPU
  - Pour un régime de parallélisme bas, le CPU enfonce le GPU
  - Un GPU n'est supérieur au CPU que pour PR > 1000 (en fait, ~1000000)
  - Le GPU n'atteint son optimum QUE pour certains PR
  - Sans une caractérisation, des déceptions à prévoir…
  - OpenCL est un bon compromis comme langage *PU
  - Python reste l'approche la plus rapide de OpenCL
- Ce qu'il faut faire : expérimenter !