Lyon Calcul 2017
Des architectures parallèles Aux programmes parallélisés

De la métrologie
À la mesure de la performance.

Loi de Brooks :

“Ne demandez pas à 9 femmes de faire en 1 mois ce qu’une peut faire en 9…
Même si au début, cela peut être plus agréable !”

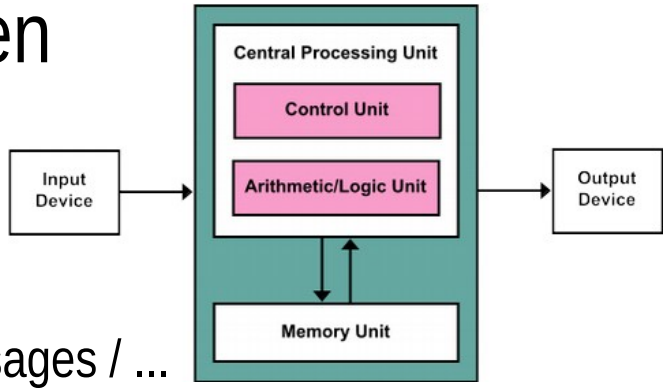
Emmanuel Quémener

Conseils avant écoute...

- Je suis le produit de l'université française il y a 25 ans
- Je ne suis pas diplomé en informatique...
  - Mais j'utilise les ordinateurs depuis 1/3 de siècle et Linux depuis 22 ans
- Je suis physicien de formation...
  - Mais j'ai travaillé sur les calculateurs analogiques durant ma thèse
- Je suis ingénieur de recherche...
  - Mais j'améliore ma connaissance de tout le spectre IT depuis 25 ans
- Mes expériences les plus significatives d'ingénieur ?
  - KISS : pour *Keep It Simple Stupid*
  - « Si vous ne pouvez pas prouver que vous avez fait le travail, ce n'est pas la peine de l'entreprendre ! »

Quelle vue du contexte ? Point de vue d'un physicien...

- Approche « système » du physicien
  - Héritage des calculateurs analogiques
  - Le « système » informatique :
    - Réseau / matériel / OS / librairies / codes / usages / ...



- Approche « Saint Thomas »
  - Apprentissage inductif par ma seule mesure
- Approche « pilote d'essai »
  - Caractérisation, recherche d'une exploitation optimale...

Précautions pour ce cours… Ce que cela ne sera pas !

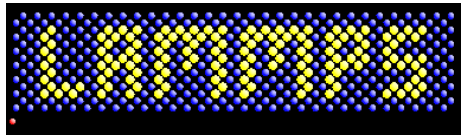
- Une introduction générale au parallélisme
  - Les cours organisés par Lyon Calcul
  - https://computing.llnl.gov/tutorials/parallel_comp/
- Une introduction aux langages de parallélisation
  - MPI : https://computing.llnl.gov/tutorials/mpi/
  - Posix Threads : https://computing.llnl.gov/tutorials/pthreads/
  - OpenMP : https://computing.llnl.gov/tutorials/openMP/
  - **C'est là que j'ai pas mal appris !**
- Je veux partager ce qui n'est JAMAIS dit (ou peu...)

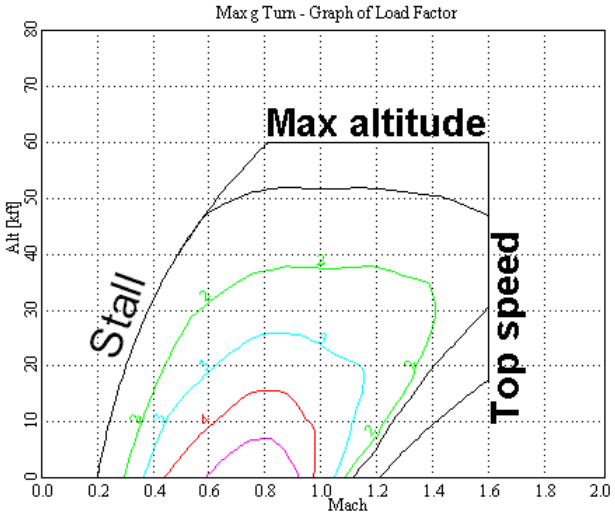
Centre Blaise Pascal ~ Dryden FR
Un petit exemple illustratif

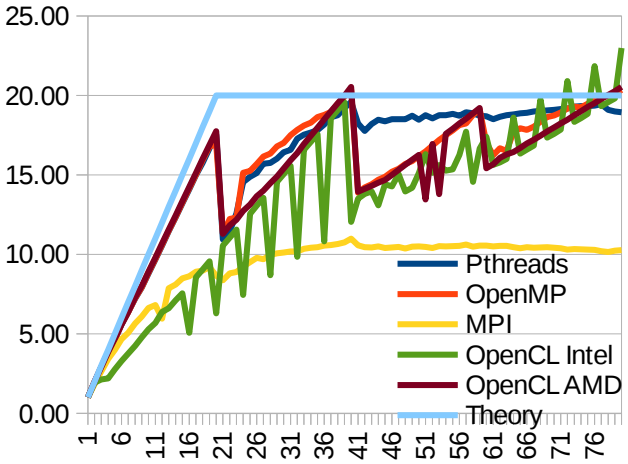
- Nasa X-29
- Cellule de F-5
- Moteur de F-18
- Train de F-16
- Etudes
  - Plans « canard »
  - Incidence >50°
  - « Fly-By-Wire »

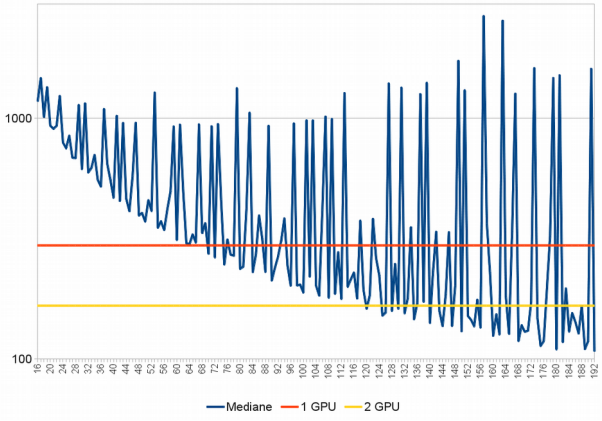
Recycler, réutiliser, explorer de nouveaux domaines...

Le but : de l'enveloppe de vol...
... aux enveloppes de parallélisme







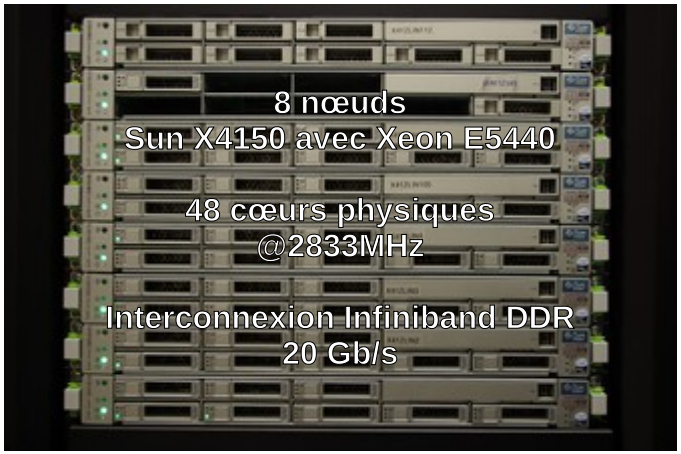


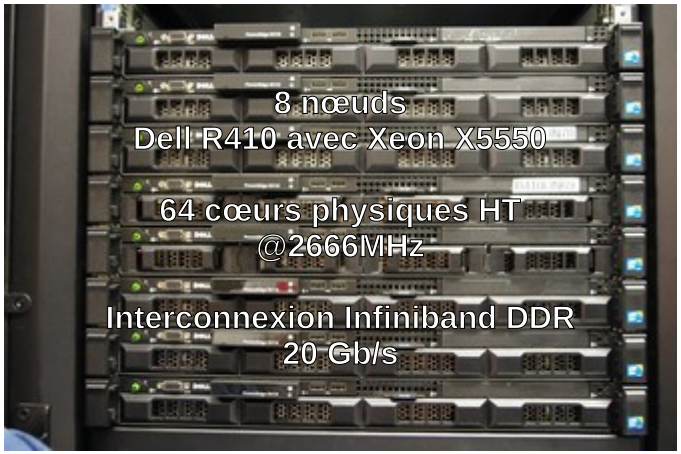
De la démarche scientifique…
… À l'expérience numérique

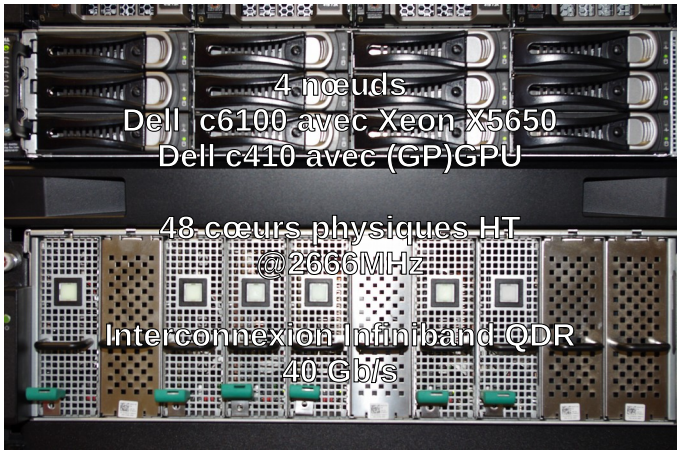
Plateforme expérimentale du CBP 10 plateaux techniques permanents

- Multi-nœuds : 5 grappes de 4 à 64 nœuds
  - Nœuds/Cœurs : 64/512, 8/64, 8/64, 4/48, 8/128
- Multi-cœurs : 30 de 2 à 28 cœurs, de 1.8 à 4.7 GHz
- (GP)GPU : 45 modèles différents de GPU (AMD & Nvidia)
- Intégration : 24 machines virtuelles en 32 et 64 bits
  - Debian de Lenny à Sid, Ubuntu 10.04 à 16.04, Centos 5.5 à 7.3
- Matériel exotique : 3 ARMv7 sous Debian Jessie ou Ubuntu
- Plateau technique 3D :
  - 2 stations, 2 vidéoprojecteurs, 20 moniteurs, 4 paires de lunettes
- COMOD : « Compute On My Own Device »
  - Même Single Instance Distributing Universal System (SIDUS)

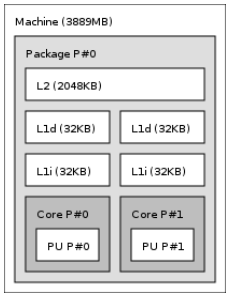
Plateau multi-nœuds : 5 «grappes» 92 nœuds, 3 réseaux spécifiques

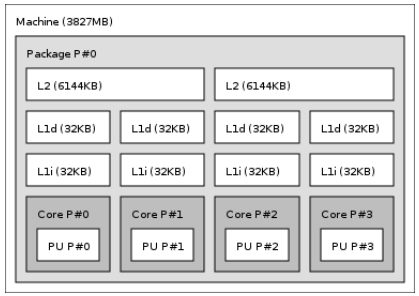


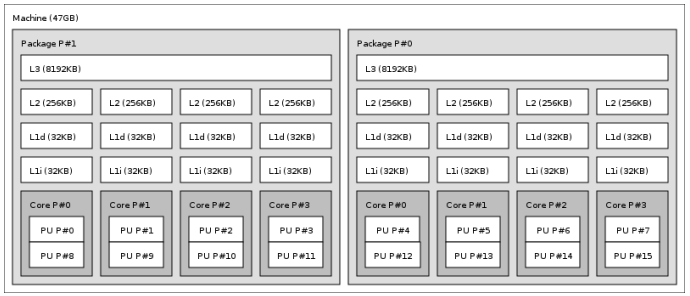


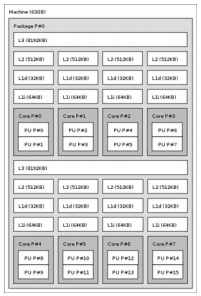


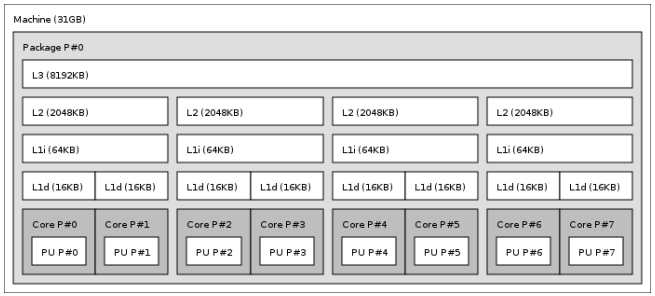
Plateau multi-cœurs de 2 à 28 cœurs : petit bestiaire...

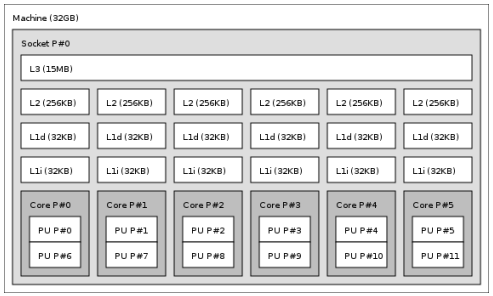


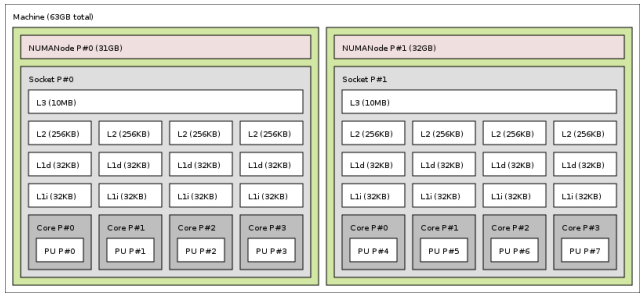


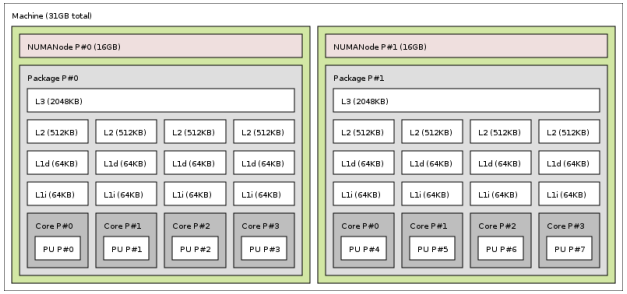


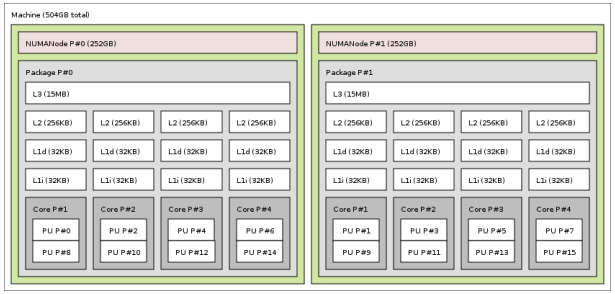












Plateau multi-shaders : (GP)GPU 45 modèles différents...

GPU Gamer : 15

- 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 1080
- Nvidia GTX 1080 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

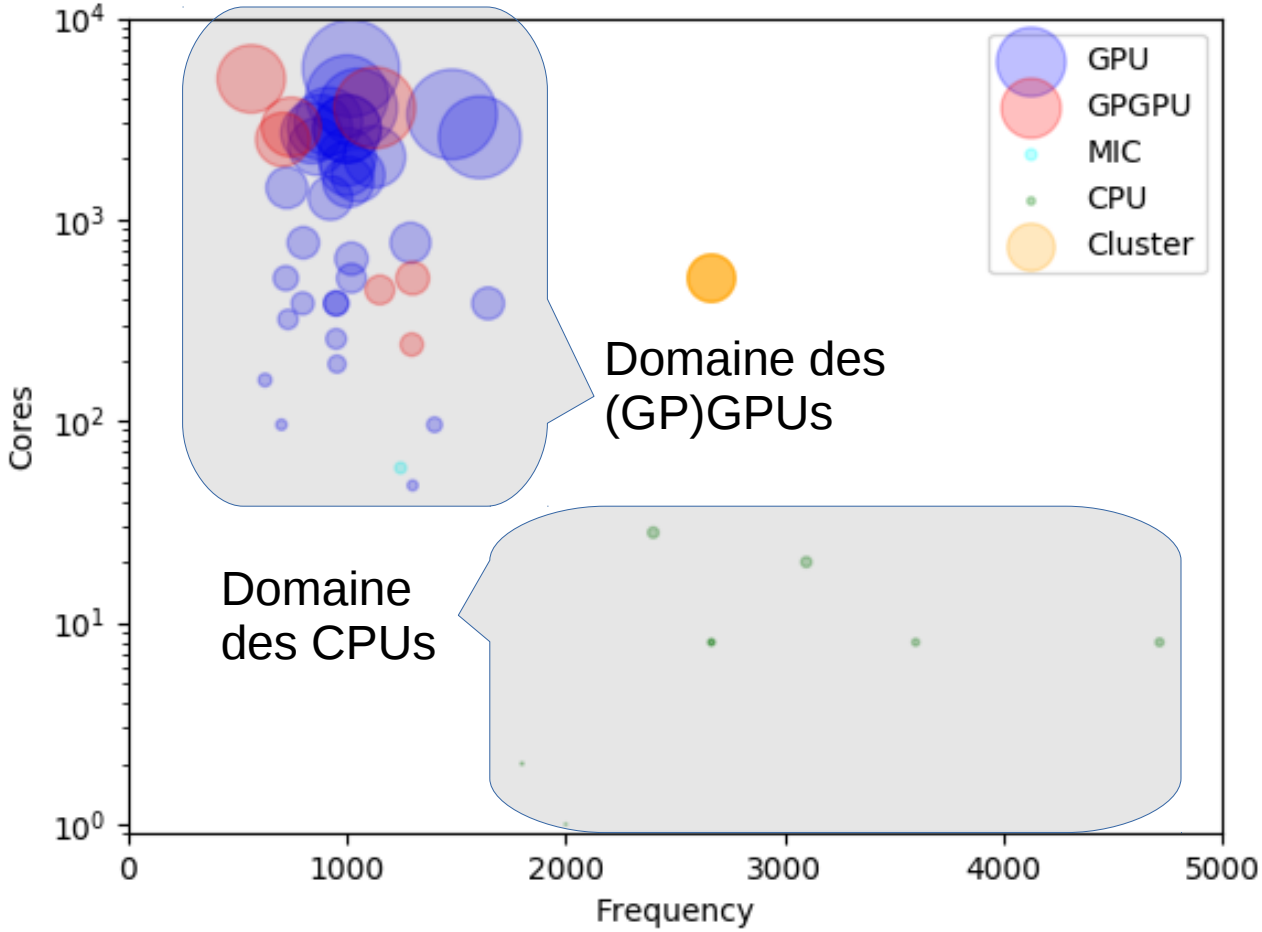
GPU desktop & pro : 11

- Nvidia Quadro 600
- Nvidia Quadro 4000
- Nvidia Quadro K2000
- Nvidia Quadro K4000
- Nvidia NVS 310
- Nvidia NVS 315
- Nvidia GT 430
- Nvidia GT 620
- Nvidia GT 640
- Nvidia GT 710
- Nvidia GT 730

GPU AMD Gamer & al : 11

- AMD Radeon HD 5850
- AMD Radeon HD 7970
- AMD R9 380
- AMD R9 290X
- AMD R9 295x2
- AMD R9 Fury
- AMD Nano Fury
- AMD Radeon HD 6450
- AMD Radeon HD 6670
- AMD R7 240
- AMD Kaveri A10-7850K

Représenter les ressources parallèles… Et les comparer !



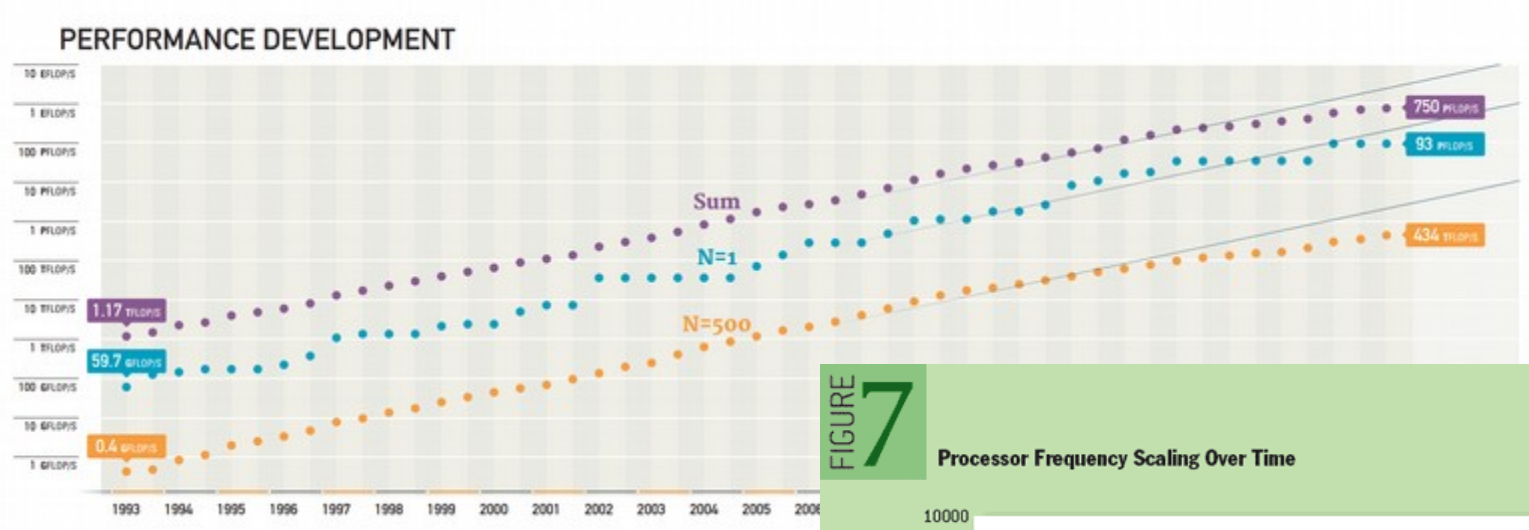
Le Parallélisme en 7 questions CQQCOQP

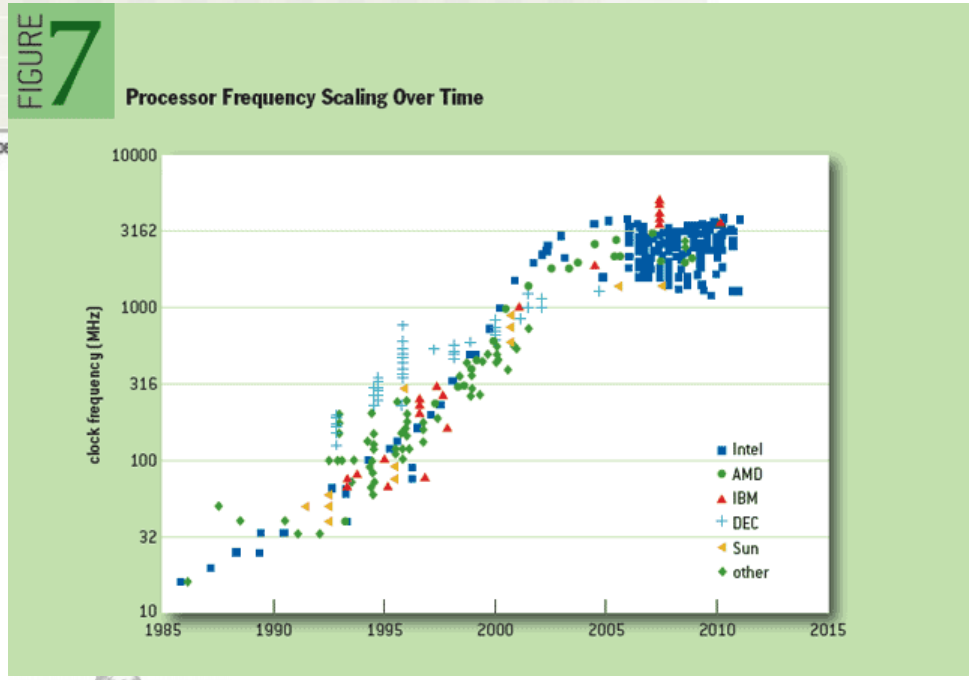
- Méthode analytique
  - Comment ? Quoi ? Qui ? Combien ? Où ? Quand ? Pourquoi ?
- Approche intéressante pour ne rien oublier
- Problème : intrication entre les questions
- Avantage : séparation des ambiguïtés
- Essayons de ne rien oublier !

Pourquoi le parallélisme : chemin… Où sommes-nous ? Où allons-nous ?

- Où sommes-nous ? Nous utilisons des codes quotidiennement
- Où allons-nous ? Nous voulons plus de « performance »
- Comment y aller ? Le parallélisme est une voie mais pourquoi ?
- Avant de « tomber dans le trou du lapin blanc » du parallélisme :
  - Quelles sont les pratiques d'utilisations ?
  - Comment définir la performance d'un code ?
  - Quel critère de performance choisir ?
  - Comment atteindre la performance souhaitée ?
  - Comment vérifier que la performance est bien atteinte ?

Un cadre historique particulier : Comment augmenter sa puissance ?



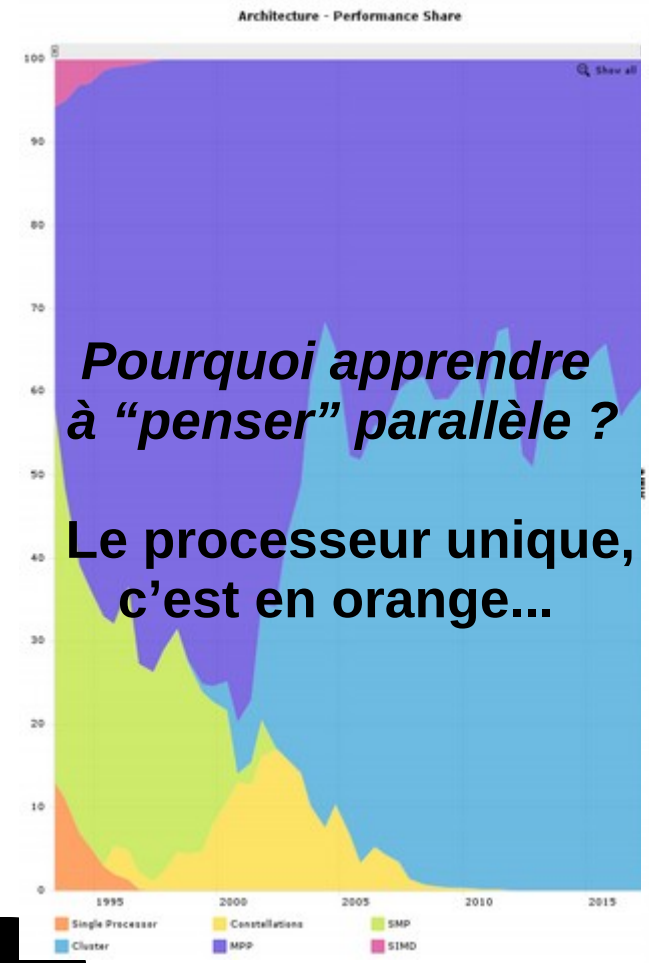


… avec un tassement

(voire une régression)

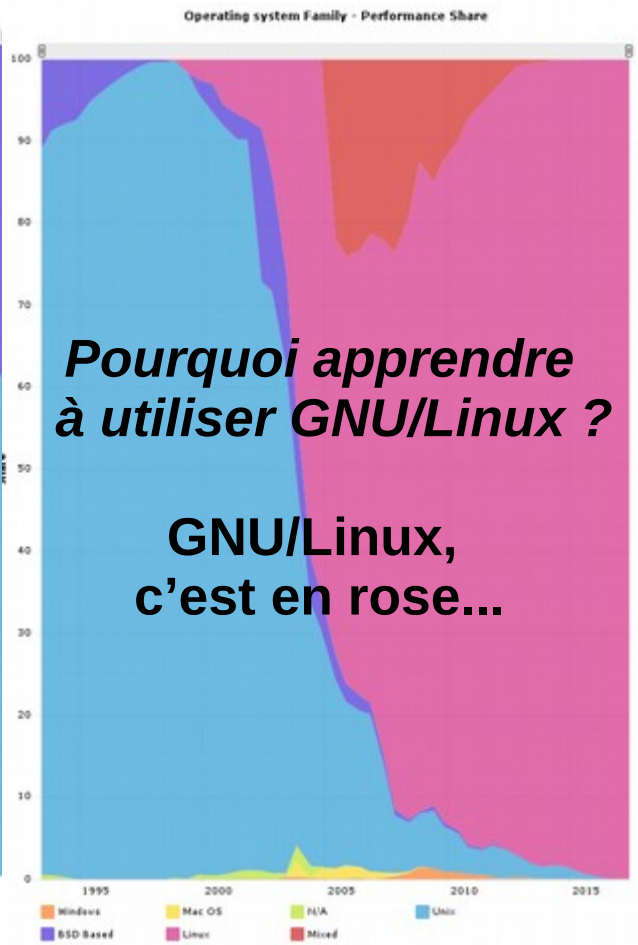
de la fréquence ?

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'1/5 le sont, mais...
½ dans le TOP10

Codes & Performance : Quelles définitions choisir ?

- Etymologie (Etymonline)
  - Code : du latin codex « livre, livre de lois »
    - « compilation systématique de lois » (1236)
    - « système de communication télégraphique » (1866)
  - Performance :
    - « accomplissement » (de quelque chose)
    - Signification « une chose réalisée » autour des années 1590
    - « ensemble de caractéristiques optimales d'un système » (1929)
- Et nous choisissons
  - Code : les deux :-)
  - Performance : les trois :-)

Si calculer, c'est cuisiner…
Le code, ce n'est que la recette...

Code ~ Recette

Ordinateur ~ Cuisine

Données d'entrée ~ Ingrédients

Données de sortie ~ Repas

Processus ~ Cuisine

Unité de contrôle ~ Cuisinier

ALU ~ Ustensile

Moi ~ Client

Requête de *batch* ~ Commande

Quelques définitions & sigles...

- ALU : Arithmetic & Logic Unit, Unité Arithmétique et Logique
- CPU : Central Processing Unit, Unité de traitement Centrale,
- Flops : Floating Point Operations Per Second, Opérations flottantes par seconde
- (GP)GPU : (General Purpose) Graphical Processing Unit, Circuit graphique
- MPI : Message Passing Interface, Interface de communication par messages
- RAM : Random Access Memory, Mémoire à accès aléatoire
- SMP : Shared Memory Processors, Processeurs à mémoire partagée
- TDP : Thermal Design Power, Enveloppe thermique
- Et quelques autres :
  - **PR** : Parallel Rate, taux de parallélisme (NP en MPI, Threads en OpenMP, Blocks, WorkItems en GPU)
  - **Itops** : Iterative Operations Per Second
  - **QPU** : Quantum Processing Unit (program exécuté avec PR=1)
  - **EPU** : Equivalent Processing Unit (PR déduit de l'optimal d'exécution du programme parallèle)

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 !

Performance : comment ?
Une question d'observables

La performance en sport

- Pour faire un 100 mètres ?
- Pour boucler un marathon ?
- Pour lancer un poids ?
- Pour accomplir un heptathlon ?

Performance : comment ? Une question d'objectifs !

- Mettre tous les bagages et la famille dans la voiture
- Attirer l'attention des filles en sortant de boîte de nuit
- Aller d'un point A à un point B dans une ville embouteillée
- Gravir Pikes Peak aux USA



Performance : Conditionnée par les objectifs

- La vitesse : Le temps écoulé (ou Wall Clock) (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 ?

Le travail comme critère de perf'

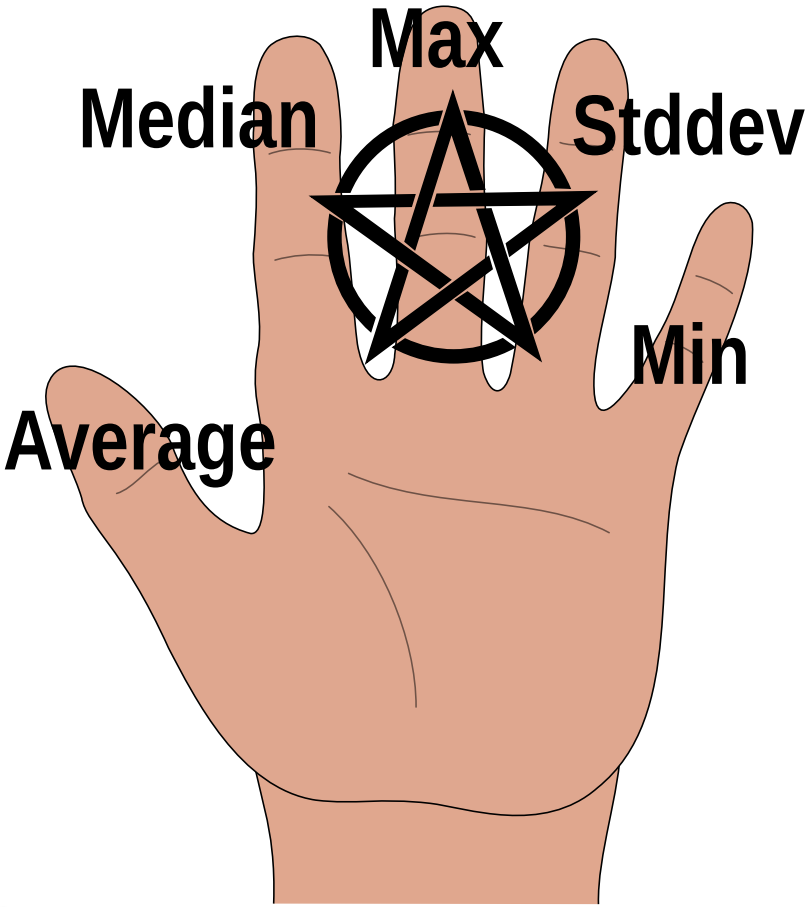
- Travail : « Time is money »
  - Ressources : CPU, RAM, GPU, stockage, réseau, ...
  - En fait, une Matriochka :
    - CPU : plusieurs cœurs, CU, ALU, piles, ...
    - RAM/SRAM : 4 niveaux
    - Stockages : local, lent & partagé (NFS), rapide & partagé (GlusterFS, Lustre, ...)
    - Réseaux : lent (Gigabit), rapide & basse latence (InfiniBand, Omnipath)
- Job : réservation (& immobilisation) de ressources
  - Classiquement, dans un système de batchs : Slots * Wall Clock
- Pour un code, quelle est l'empreinte système
  - Outils de profilage, outils système

La scalabilité comme critère

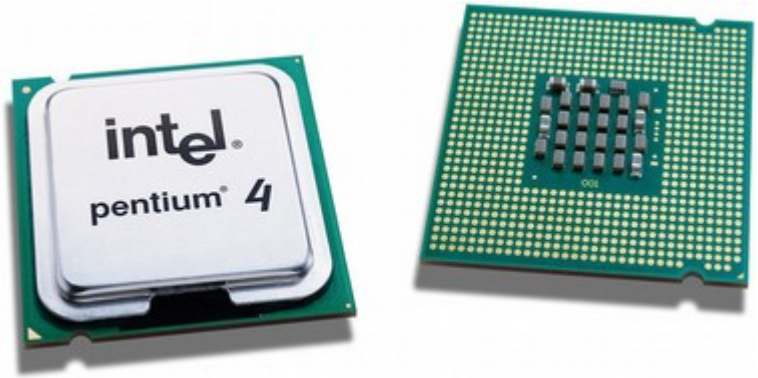
- La scalabilité :
  - Dans une tâche à accomplir : Temps écoulé ? f(Temps écoulé)
  - Dans les ressources à mobiliser : g(Ressources Système)
- Pièges à éviter :
  - Les effets d'échelle (en fait, les effets de seuil sont pire)
  - Besoin d'un chef d'orchestre ? Du quatuor à l'orchestre symphonique…
  - Quelle que soit le programme, les ressources machines sont limitées…
    - Vous pensez vraiment que ça me fait rire :-/ ?
- **La parallélisation incontournable, mais pourquoi ?**

Mesure de Performance & Scalabilité Pen(s)tacle des statistiques :

- Pourquoi améliorer ses stats' ?
  - Parce que vous faites de la science !
- Le pentacle de la statistique
  - **Moyenne** : le premier, mais le pire
    - Processus d'initialisation, aléatoires, ...
  - **Médiane** : celui à préférer (le plus naturel)
  - **Max** : le meilleur ou le pire
  - **Stddev** : indicateur de la dispersion
  - **Min** : le pire ou le meilleur
- **Variabilité** : ratio Stddev/Médiane

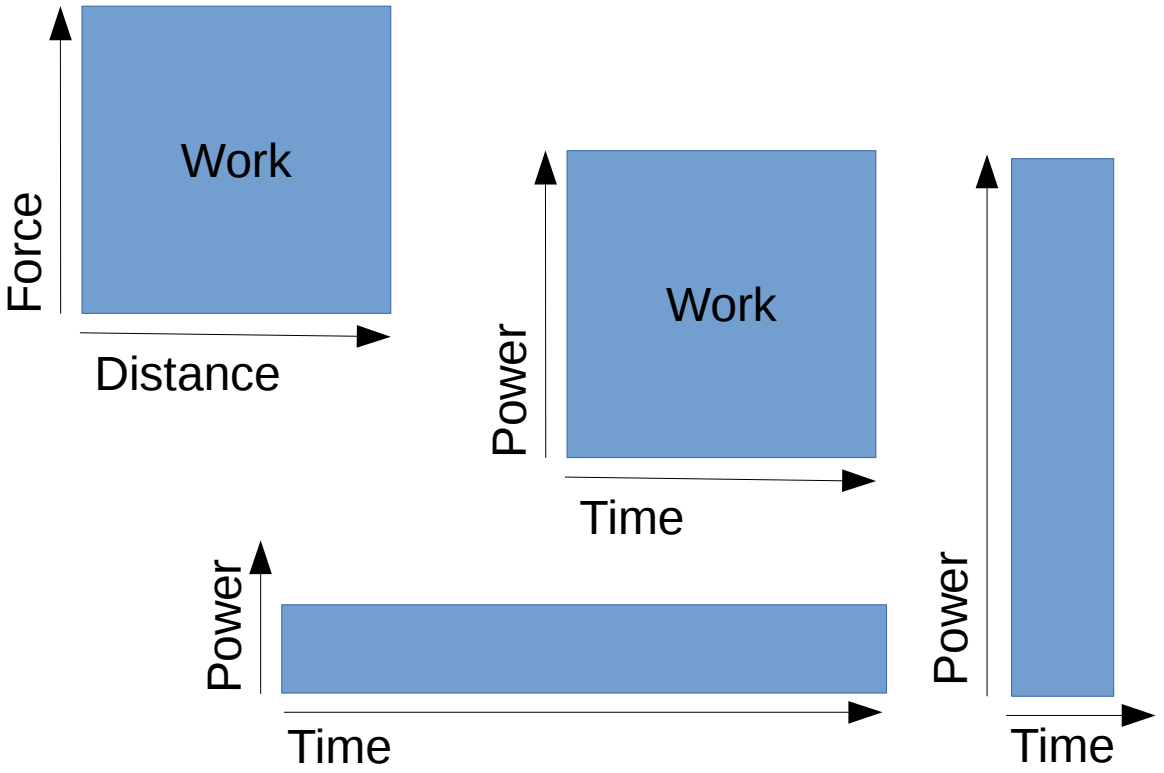


Energie dans les ordinateurs… La lorgnette du physicien...



Électronique & Thermique

Energie en calcul scientifique… Le point du vue du physicien



Mécanique

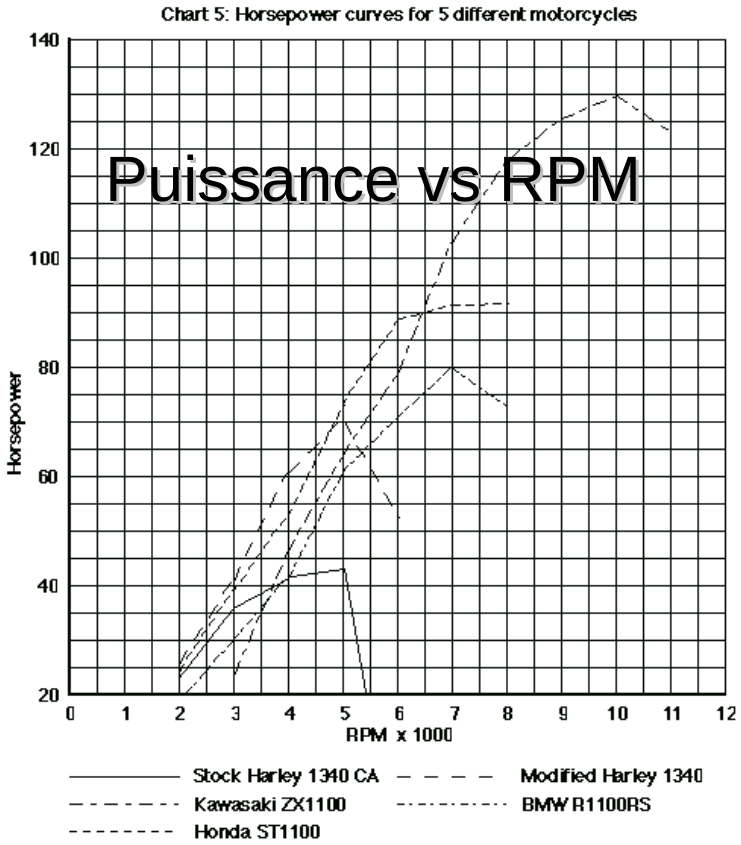
La puissance est définie
Comme le produit de :

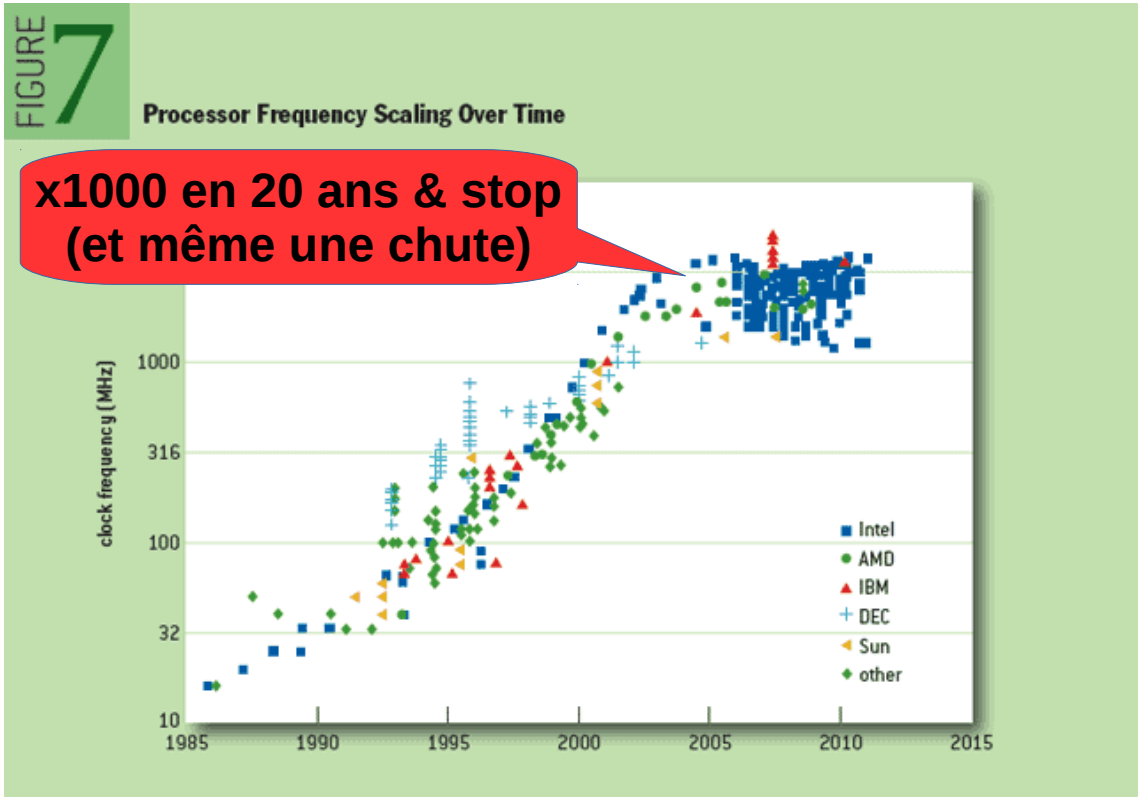
- Fréquence
- Nombre d'unités
- Puissance unitaire

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

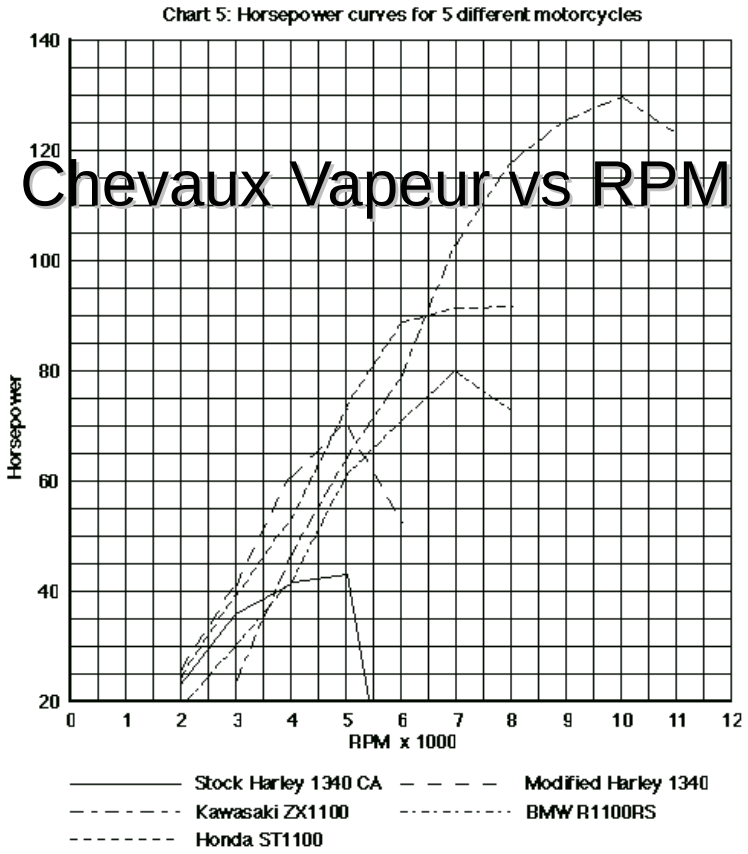
Caractérisation des « moteurs » de traitement...

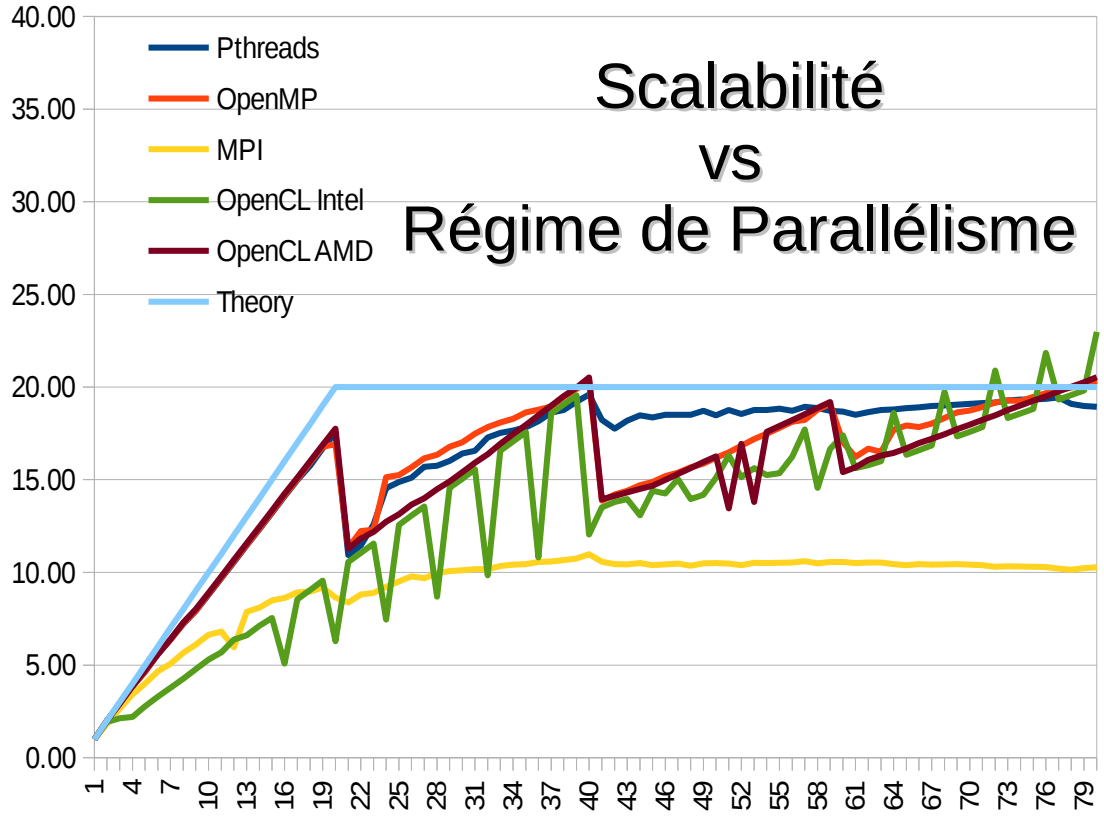
Il y a longtemps, entre 1985... et 2005, la variable est la fréquence !





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

Pourquoi le parallélisme (est inévitable) ? Et sa contrainte est la TDP

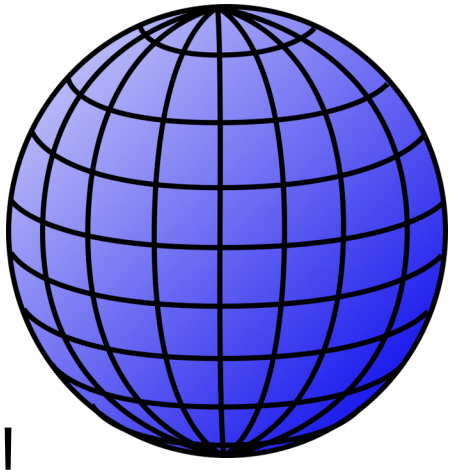
- Grandeur et décadence de la fréquence
  - Entre 1981 et 1999 : de 4 MHz à 400 MHz x100 en ~20 ans
  - Entre 1999 et 2004 : de 400 MHz à 3 GHz x~10 in 5 years
  - Entre 2004 et 2009 : de 3 GHz à 2 GHz
- Thermal Design Power : enveloppe thermique de dissipation maximale
- TDP = ½ C V^2 f
  - C = Capacitance, f = fréquence, V = tension d'alimentation
- TDP pour un processeur : 150 W (sur 4 cm^2)
  - Densité de chaleur d'une plaque à induction !
- TDP devient le facteur limitant de puissance

Capacitance = Finesse² . Nb Transistors . Constante de Mylq (~ 0.015)

Solution viable : augmenter les unités de traitement (PU)

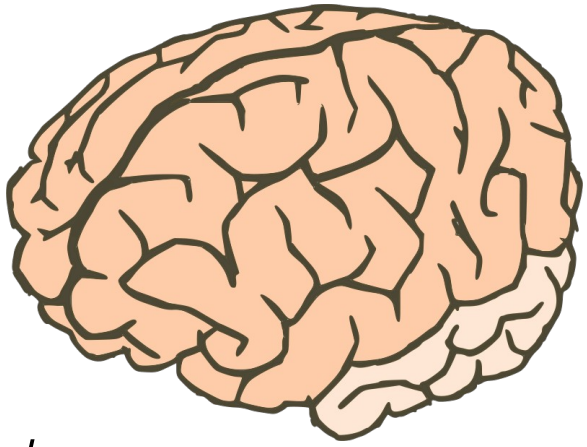
Qu'est-ce que le parallélisme ? Retournons à la source

- Etymologie (etymonline.com) : à côté d'un autre
  - De para- « à côté »
  - De allelois « chacun », de allos « autre »
- Parallélisme : tâches à accomplir, ressources limitées
  - Exécution de différentes tâches en parallèles
  - Exécution d'une tâche sur plusieurs ressources
    - Communications éparses : gros grain
    - Communicatons fréquentes : grain fin
- Paradoxe du parallélisme, des méridiens !



Où est le parallélisme ?

- Où se trouve le meilleur ordinateur ?
  - Entre vos oreilles !
  - De 20 à 200 milliards de neurones
  - De 125 à 220 milliers de milliards de synapses
  - Capacité de calcul (IBM) : 36 Pflops, 3.2 Pbytes :-/

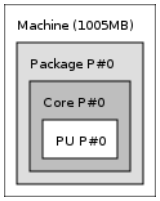


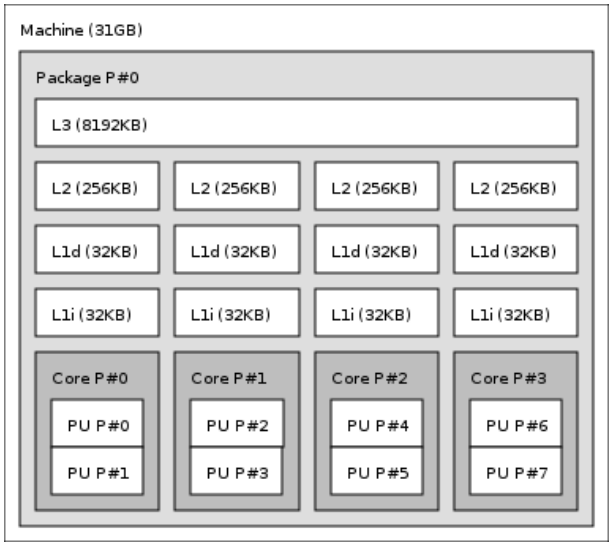
- Dans le « must » des GPGPU :
  - 3584 ALUs
  - 16 GB et 720 GB/s de bande passante
  - Capacité de traitement : 9.3 Tflops
  - En fait, en tirer 4 Tflops (FP64), c'est exceptionnel !

Combien pour le parallélisme ? Des multi-cœurs aux myri-ALUs

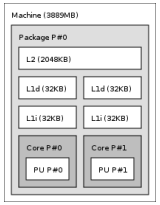
- CPU, les cœurs comme unité...
  - 4 dans un portable,
  - 16 dans une station,
  - 48 dans un nœud
- Des GPU aux GPGPU :
  - Une petite carte graphique : 128 ALU, 512 MB de RAM
  - Une grosse carte graphique : 4096 ALU, 8 GB de RAM
- Une grosse carte GPGPU : 3584 ALU, 16 GB de RAM
- Un accélérateur Xeon Phi : 61 CPU (des unités Pentium de 1995)

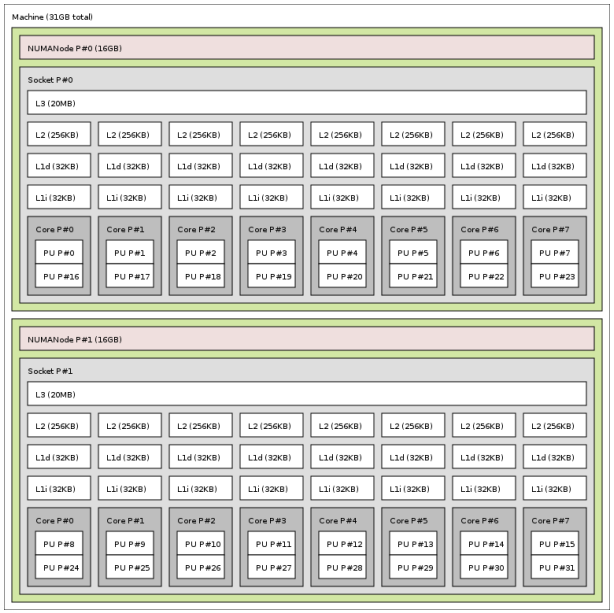
Evolution des machines…
De 2004 à 2013, sur les laptops !



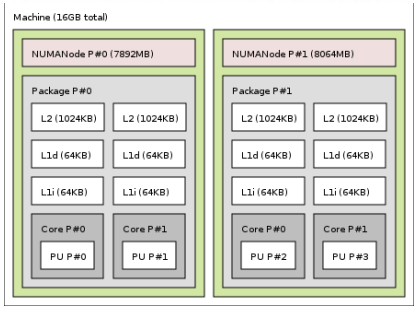


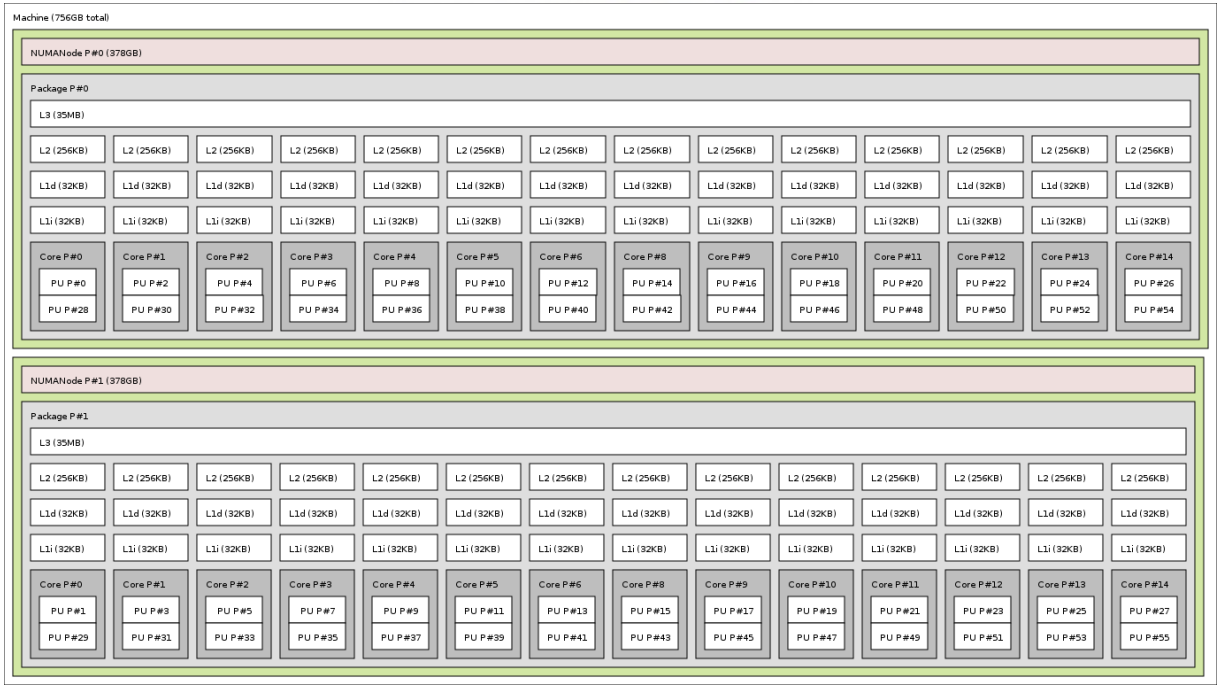
Evolution de 2007 à 2016
Sur les stations de travail



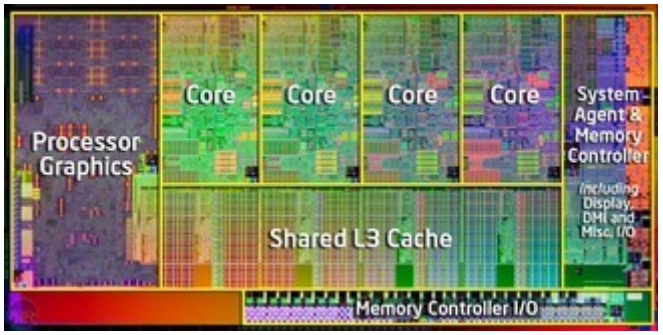


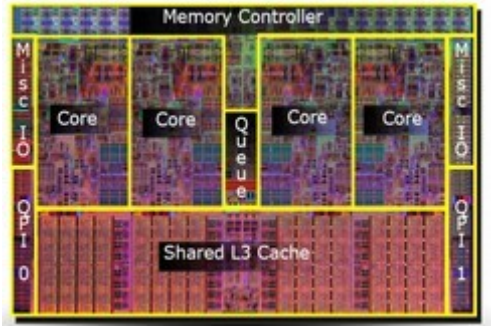
Evolution de 2007 à 2016 sur des serveurs...

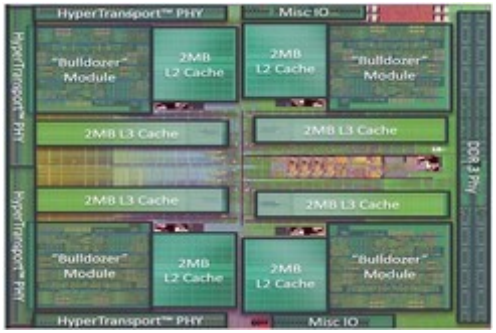


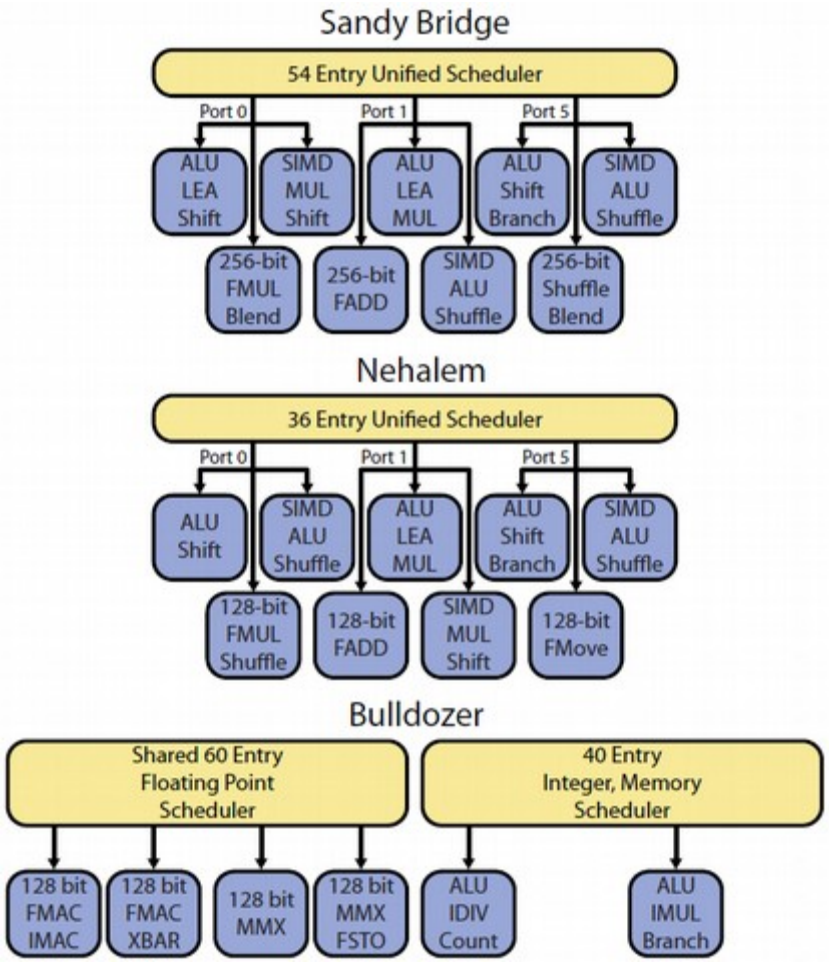


A l'intérieur d'un socket
Exemples de 3 quadri-cœurs

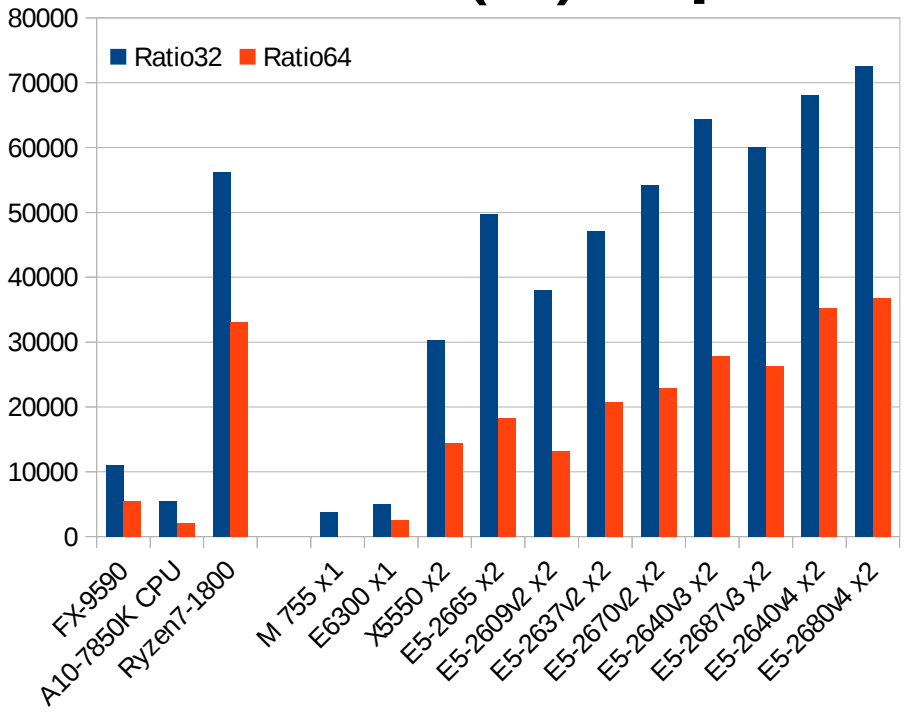




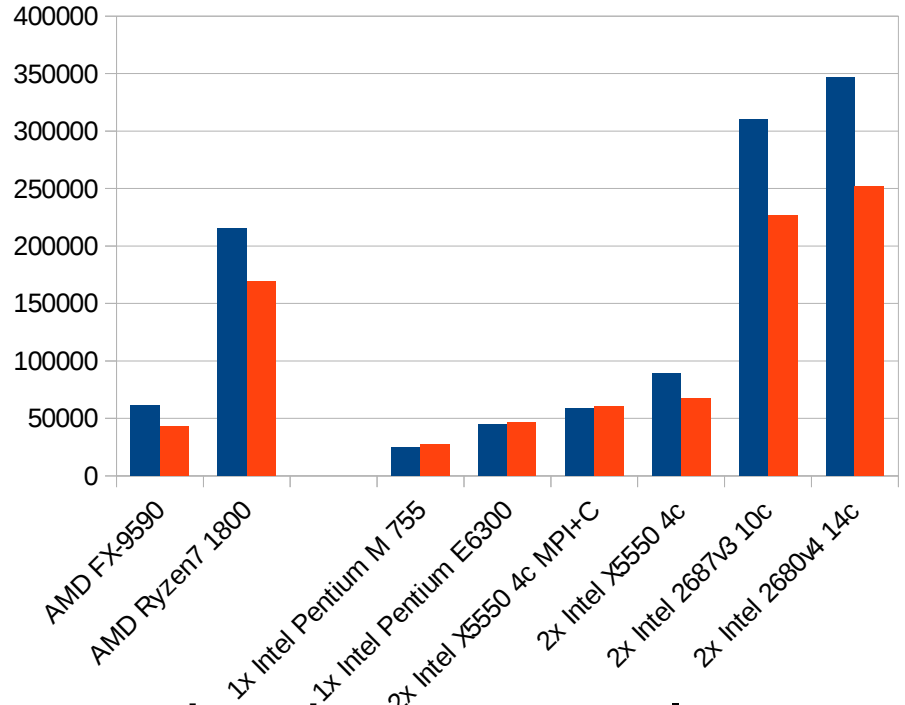




Renormalisation de Performances sur CPU(s) : par cœur, par MHz



Code de N-Corps



Code Pi Monte Carlo

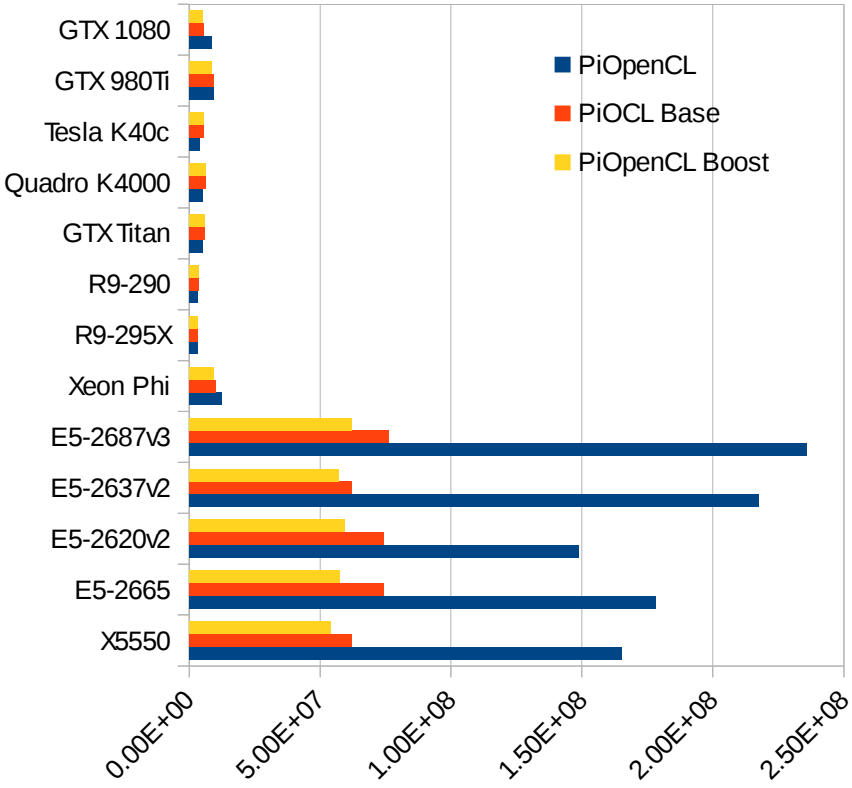
Oui, la puissance augmente avec le temps,
Mais seulement sur codes vectorisés

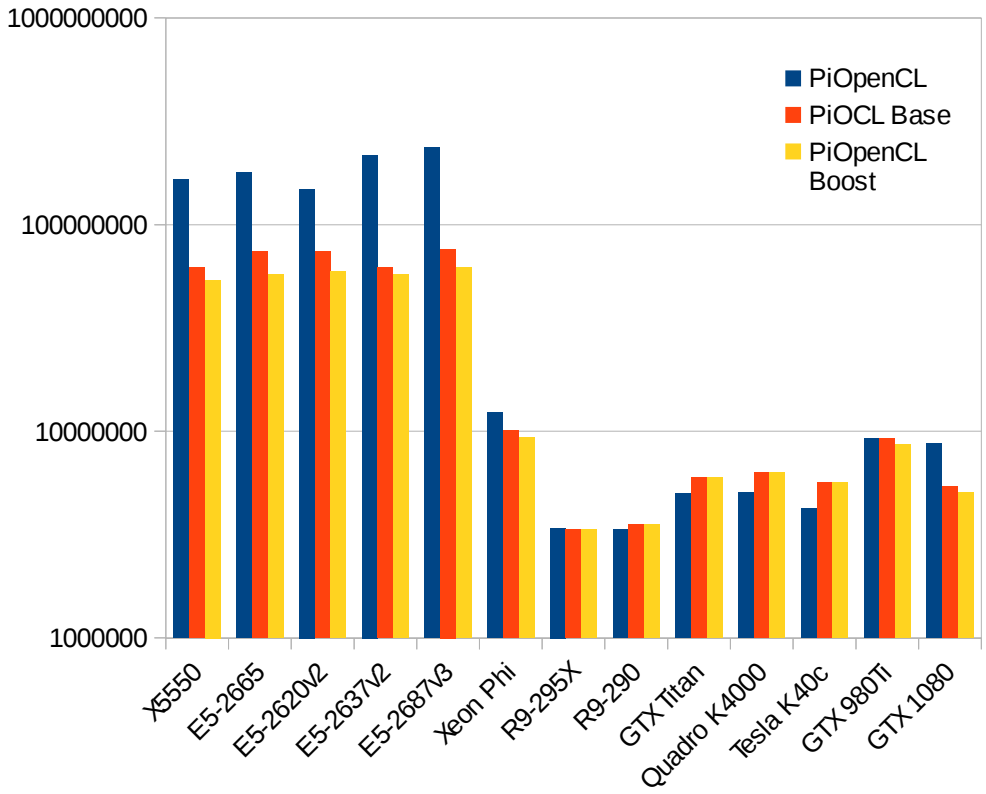
Combien pour le Parallélisme ? Temps, Silicium, Complexité...

- The 3 coûts temporels :
  - Coût d'entrée, coût d'exploitaion, coût de sortie
  - Un temps d'exécution à comparer avec le temps d'adaptation
- Silicium : les technologies ont des coûts très différents
  - SMP (Shared Memory Processors) : chères et limitées
  - MPP (Massively Parallel Processing) : exigeantes en réseaux très spécifiques
  - Les clusters sont facilement extensibles
- Complexité : corollaire d'un large nombre de portes logiques
  - Un « cœur » GPU (QPU) est plus simple qu'un cœur CPU
  - Un « cœur » GPU (QPU) est jusqu'à 50 fois plus lent qu'un cœur CPU

Pour les programmes séquentiels, le CPU (vieux et al) écrase le GPU!

- De 20 à 50x plus lent !





- 1.5 Ordre de grandeur...

Comment penser « parallélisme »
« Le grain, le problème, c'est le grain... »

- 1 Entrée / 1 Processus ? Optimisez le processus !
- 1 Entrée / Y Processus ? Optimisez chaque processus !
- X Entrées / 1 Processus ? Optimisez la distribution !
- X Entrées / Y Processus ? Optimisez les deux !

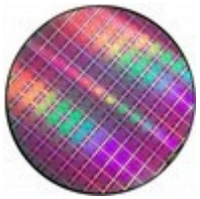
Le Grain est défini par le taux de communication !

- Grain fin : communications fréquentes (~ >> 1/seconde)
- Gros grain : communications éparses (~ < 1/seconde)
- « Embarrassing parallelism » : tâches indépendantes

Comment penser le ParallelisMe
La Taxonomie de Flynn

- SISD : Simple Instruction Simple Data
- SIMD : Simple Instruction Multiple Data
  - Vectorisation

- MISD : Multiple Instructions Simple Data
  - Pipelining
- MIMD : Multiple Instructions Multiple Data



Jetons un petit regard

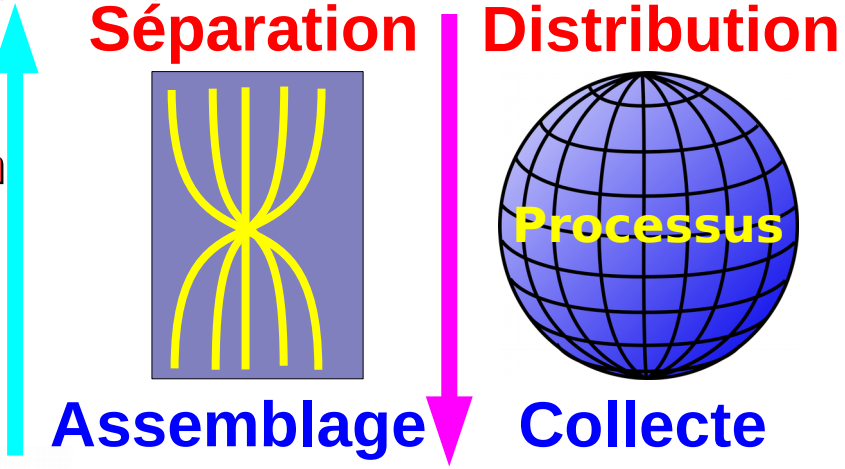
« Derrière la porte de la cuisine » (In Silicon) ?

Comment exploiter le parallélisme ? Stratégies d'exécution parallèle...

- Pipelining à grain fin, sur le silicium :
  - 5 instructions simples à la fois
    - Récupération de l'instruction
    - Décodage de l'instruction
    - Exécution
    - (Accès si nécessaire à la mémoire)
    - Ecriture du résultat
  - 2 spécifications du RISC : 1 instruction/cycle, usage de registres

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

- 2 approches différentes :
  - Vectorisation : Assemblage/Processus/Séparation
  - Distribution : Distribution/Processus/Collecte
- En fait, médianisation ;-)

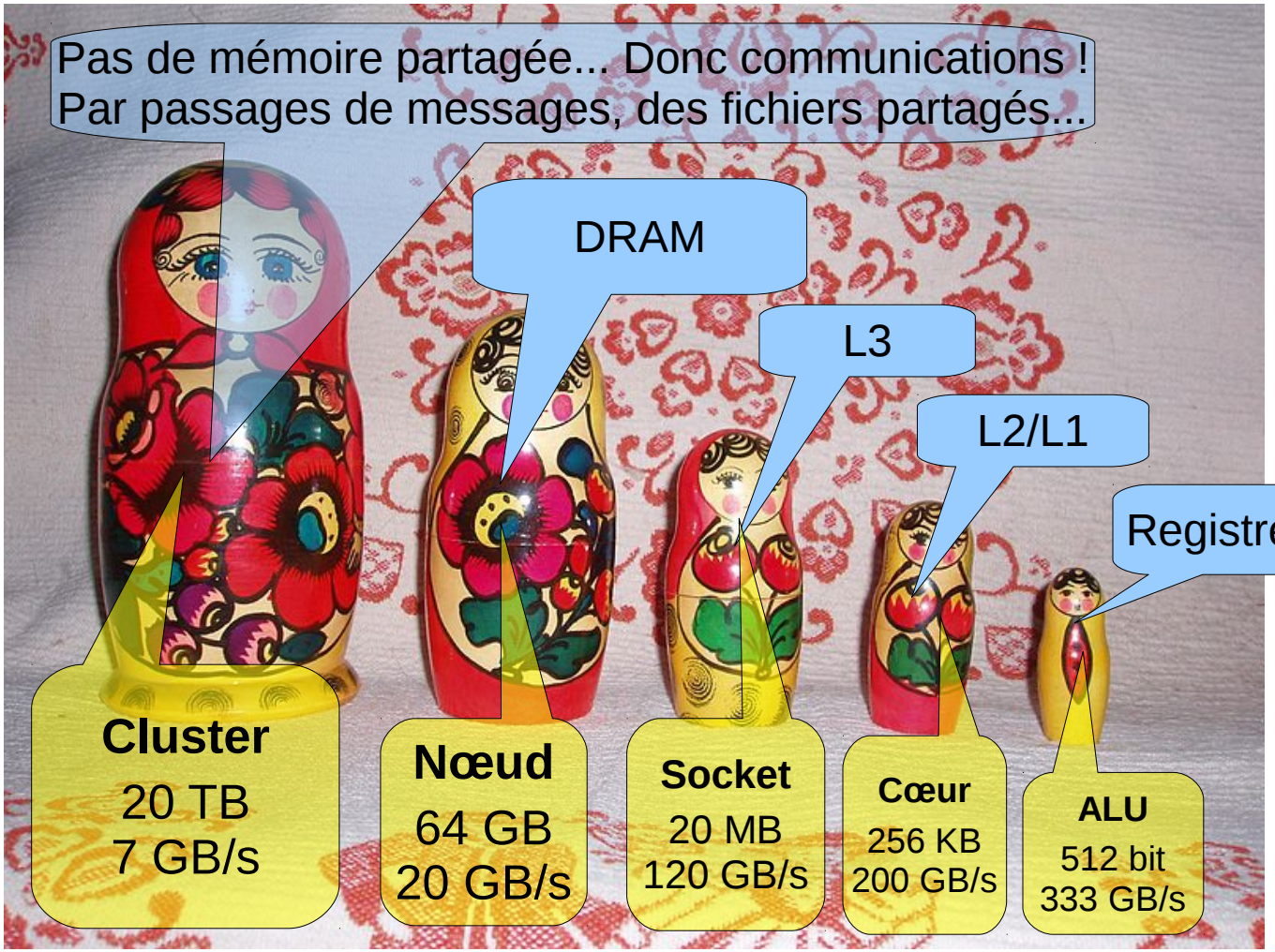


Au delà de la cuisine
Le déménagement.

- Objectif : déménager des 420 cartons de A à B
- Moyens :
  - 6 personnes pour chacune porter 1 carton
  - 2 chariots pouvant porter 6 cartons
  - Un ascenseur de chaque côté pouvant accueillir 12 cartons
  - Une camionnette pouvant contenir 100 cartons
- Comment organiser le déménagement ?
  - Quelles sont les unités parallèles, quelle est leur nature ?
  - Où retrouve-t-on les éléments de Flynn ?

Le point de vue des unités de traitement

Mémoires hiérarchiques !



Si calculer était cuisiner...
Et pour la mémoire...

Code ~ Recette

Ordinateur ~ Cuisine

Données d'entrée ~ Ingrédients

Données de sortie ~ Plat cuisiné

Processus ~ Préparation

Unité de contrôle ~ Cuisinier

ALU ~ Ustensile

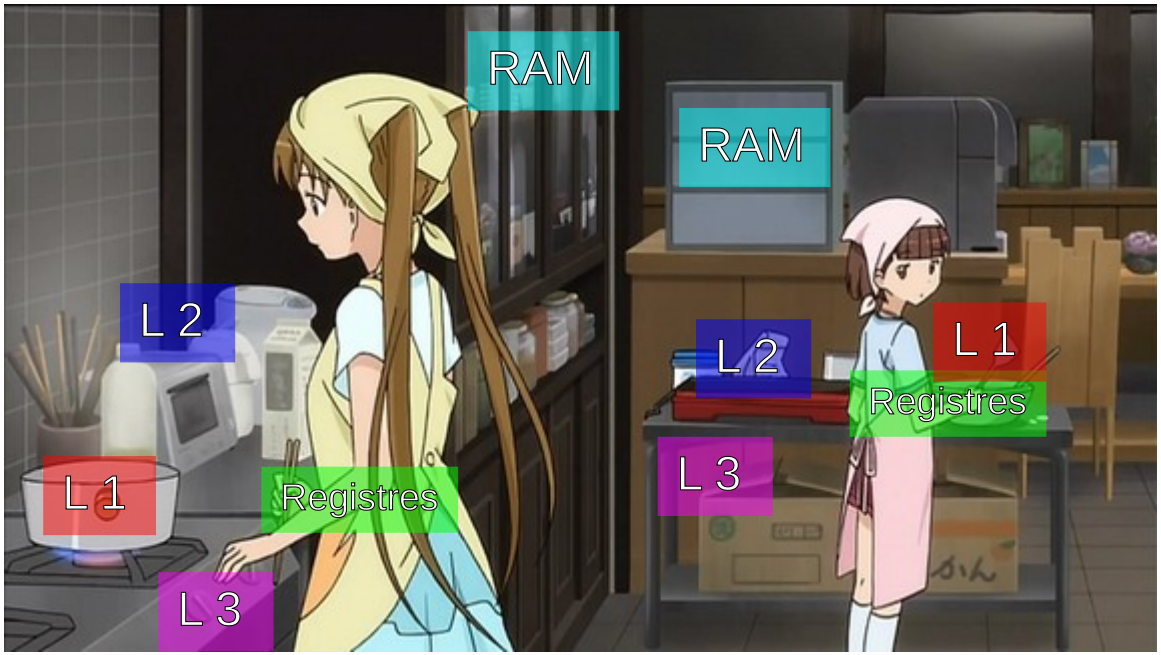
Dynamic RAM ~ Placards, tables, ...

L3 Cache ~ Tout plan de travail

L2 Cache ~ Plan de travail à un pas

L1 Cache ~ Plan de travail, récipient

Registres ~ Mains de la cuisinière



Comment programmer en // ?
Les différentes approches

Les modèles de programmation parallèles

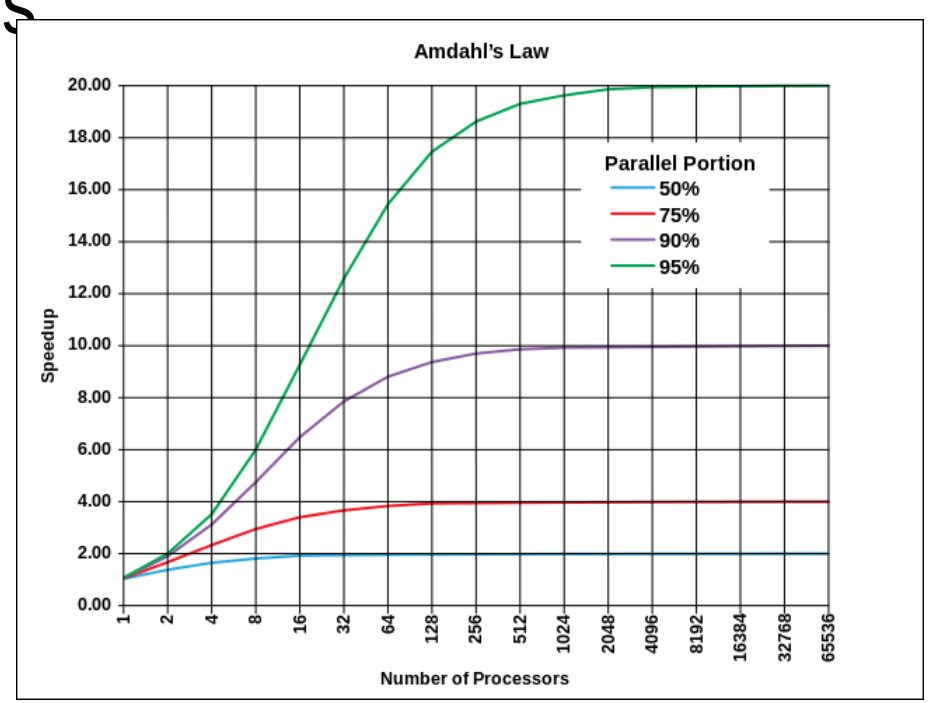
	Cluster	Nœud CPU	Nœud GPU	Nœud Nvidia	Accélerateur
MPI	Yes	Oui	Non	Non	Oui*
PVM	Oui	Oui	Non	Non	Oui*
OpenMP	Non	Oui	Non	Non	Oui*
Pthreads	Non	Oui	Non	Non	Oui*
OpenCL	Non	Oui	Oui	Oui	Oui
CUDA	Non	Non	Non	Oui	Non
TBB	Non	Oui	Non	Non	Oui*

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

Comment estimer l'efficacité // ? Loi d'Amdahl, ordre (et décadence)

- Dans le processus, 2 parties
  - Part séquentielle, en fraction s
  - Part parallèle, en fraction p
  - Temps écoulé : $T_N=T_1(s+p/N)$
  - Accélération : 1/(1-p+p/N)
  - Efficacité : 1/N(1-p+p/N)
- Accélération (& efficacité) :
  - 2 systèmes : N=500 & N=1000
  - 4 cas : 90 %, 99 %, 99.9 %, 99.99 %



Comment estimer l'efficacité parallèle ? Loi d'Amdahl, ordre (et décadence)

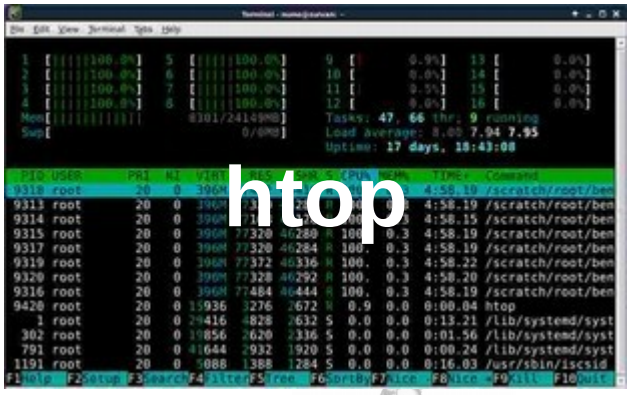
- Accélération (& efficacité) : N=500 & N=1000

Parallel Rate	N=500		N=1000	
Part parallèle	Accélération	Efficacité	Accélération	Efficacité
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%

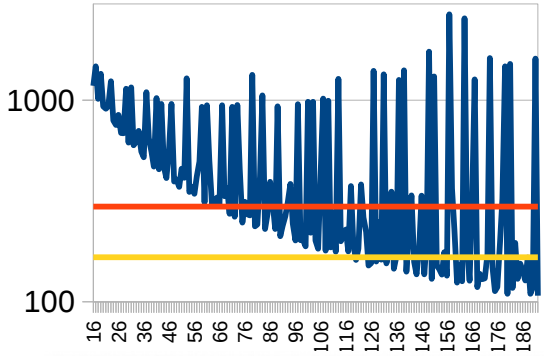
- Questions :
  - Quelle est la scalabilité de mon code ?
  - La loi d'Amdahl est-elle représentative des applications réelles ?

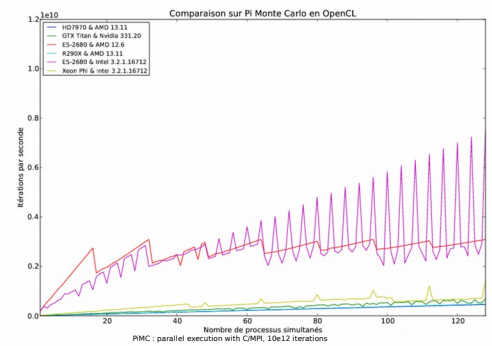
Votre « permis de conduire » en calcul scientifique ;-) ?

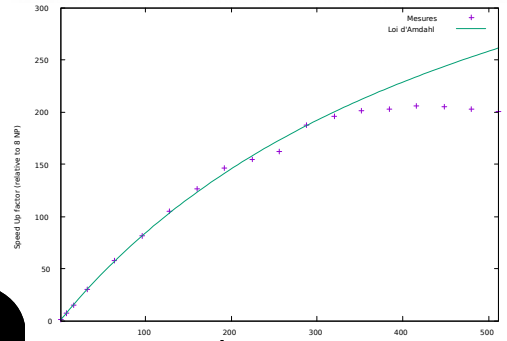
- Dans un livre français de mathématiques appliquées
  - « Les physiciens ont un usage des mathématiques que les mathématiciens assimilent facilement à de l'inconscience ! »
- Comme un BOFH de ressources informatiques
  - « Les scientifiques ont un usage typique des ressources informatiques que j'assimile aisément à de l'inconsistance ! »
- Est-ce que vous « conduisez » vos usages ?

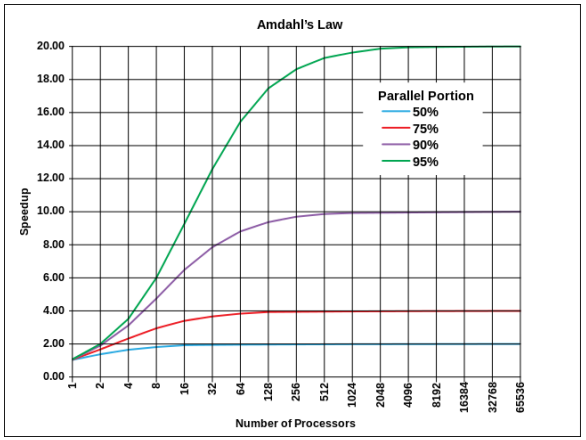


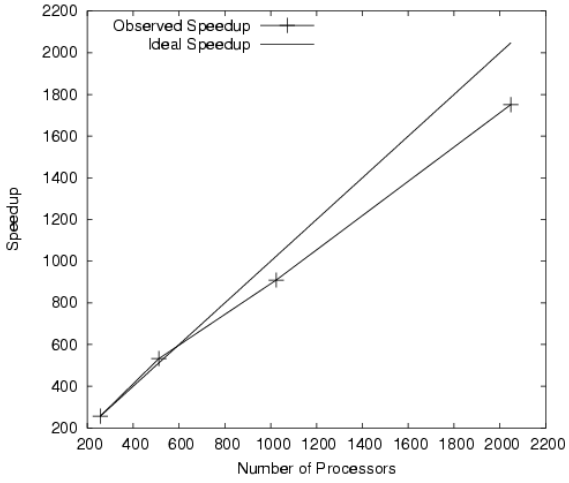
Loi d'Amdahl ? Vérité ou mensonge Prêt à prendre la « pilule rouge » ?











L’observable en informatique...

- Observable = fonction(code,données,backoffice)
  - Qu'est-ce que je maîtrise ?
- Le code ?
  - Tout le code ?
- Les données ?
  - L'accès aux données
- Le back-office
  - Réseau ? Systèmes ?
- Une constante : observer perturbe...
  - Out-of-code : time et al, mais aussi le reste
  - In-Code : commande système avec timer

Les deux temps... time & /usr/bin/time

- 2 versions de « Time »
  - Dans le terminal : build in time
    - time sleep 10
    - real 0m10.003s
    - user 0m0.000s
    - sys 0m0.000s
  - Commande : /usr/bin/time
    - /usr/bin/time sleep 10
    - 0.00user 0.00system 0:10.00elapsed 0%CPU (0avgtext+0avgdata 640maxresident)k
    - 0inputs+0outputs (0major+207minor)pagefaults 0swaps
- Exploiter /usr/bin/time, MAIS mieux !!!

/usr/bin/time : le minimum...

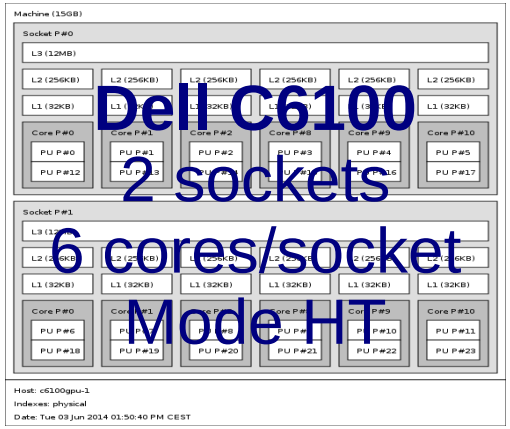
La définition de la variable TIME

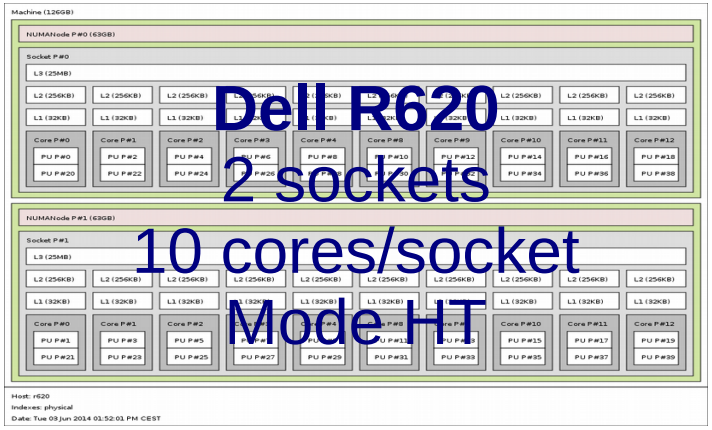
- export TIME="
  - TIME Command being timed: "%C"
  - TIME User time (seconds): %U
  - TIME System time (seconds): %S
  - TIME Elapsed (wall clock) time : %e
  - TIME Percent of CPU this job got: %P
  - TIME Average shared text size (kbytes): %X
  - TIME Average unshared data size (kbytes): %D
  - TIME Average stack size (kbytes): %p
  - TIME Average total size (kbytes): %K
  - TIME Maximum resident set size (kbytes): %M
  - TIME Average resident set size (kbytes): %t
  - TIME Major (requiring I/O) page faults: %F
  - TIME Minor (reclaiming a frame) page faults: %R
  - TIME Voluntary context switches: %w
  - TIME Involuntary context switches: %c
  - TIME Swaps: %W
  - TIME File system inputs: %I
  - TIME File system outputs: %O
  - TIME Socket messages sent: %s
  - TIME Socket messages received: %r
  - TIME Signals delivered: %k
  - TIME Page size (bytes): %Z
  - TIME Exit status: %x"

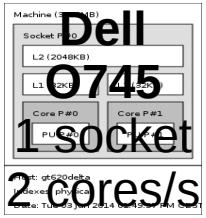
Petit exemple d'exécution

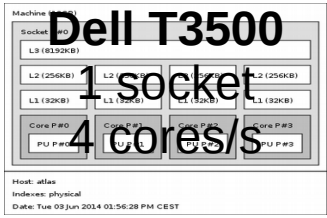
- TIME Command being timed: mpirun -np 128 ./Pi_MPI_FP32_MWC 1000000000
- **TIME User time (seconds): 44.16**
- **TIME System time (seconds): 8.84**
- **TIME Elapsed (wall clock) time : 8.14**
- **TIME Percent of CPU this job got: 651%**
- TIME Average shared text size (kbytes): 0
- TIME Average unshared data size (kbytes): 0
- TIME Average stack size (kbytes): 0
- TIME Average total size (kbytes): 0
- **TIME Maximum resident set size (kbytes): 85116**
- TIME Average resident set size (kbytes): 0
- TIME Major (requiring I/O) page faults: 307
- TIME Minor (reclaiming a frame) page faults: 243144
- **TIME Voluntary context switches: 1184545**
- **TIME Involuntary context switches: 903070**
- TIME Swaps: 0
- **TIME File system inputs: 0**
- **TIME File system outputs: 304296**
- TIME Socket messages sent: 0
- TIME Socket messages received: 0
- TIME Signals delivered: 0
- TIME Page size (bytes): 4096
- TIME Exit status: 0

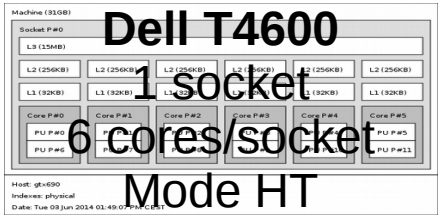
Bienvenue, Amdahl dans le monde réel !
10 machines de 2 à 40 cœurs...

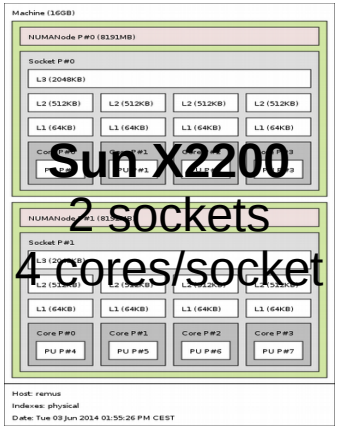


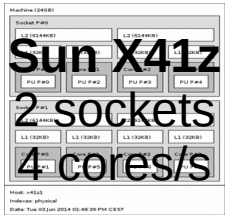




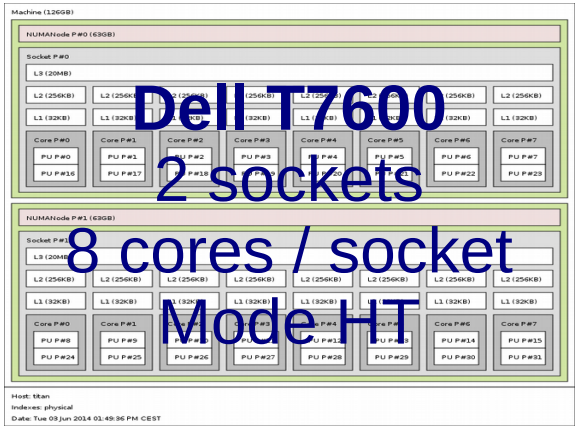


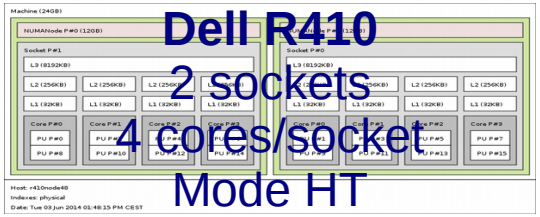


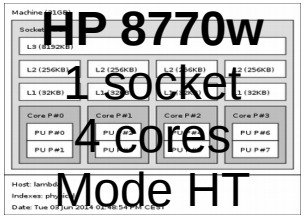




hwloc-ls comme commande

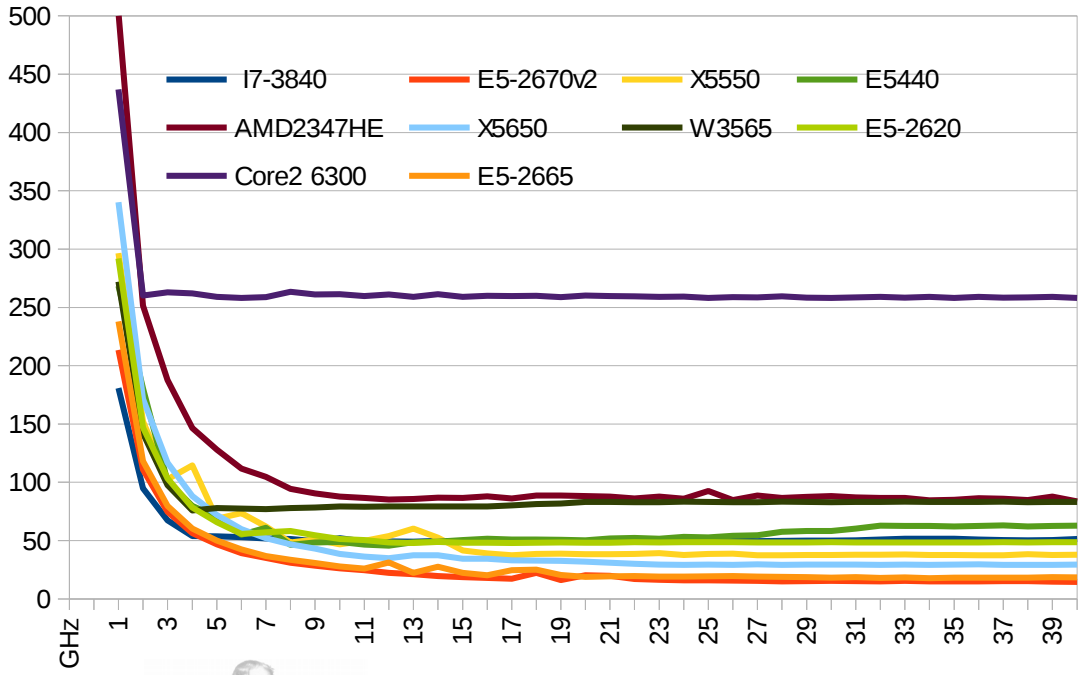






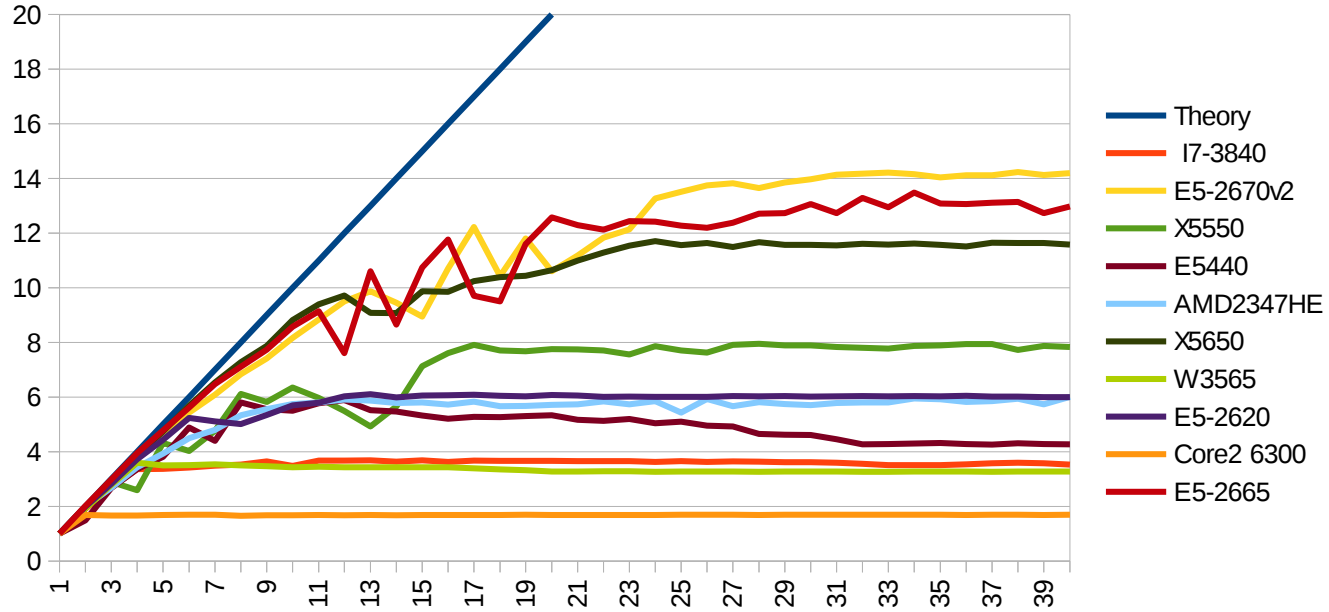
Loi d'Amdahl dans le monde réel
Un banc de test : 10 CPUs, 1 code

- Dans un nœud, 2 processeurs :
- 10 CPUs différents, de 2 à 20 cœurd (40 en HT)
- 1 application : pbzip2 (parallel bzip2)
- 1 donnée d'entrée : film encodé (1.4 GB) (pire scénario)
- De 1 processus to 80 processus
- Outil de métrologie : temps
- Observable : temps écoulé

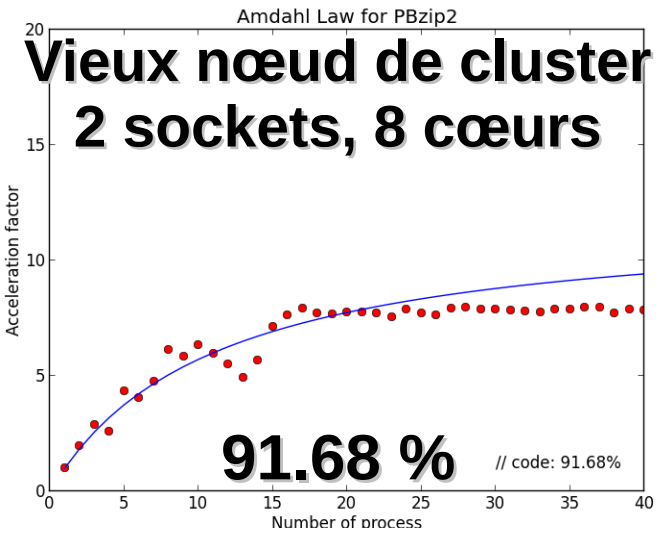


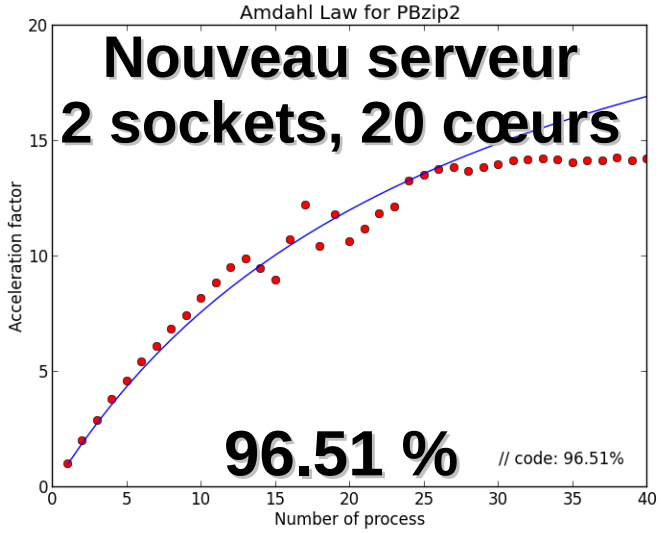
Loi d'Amdahl dans le monde réel Accélération & Variations

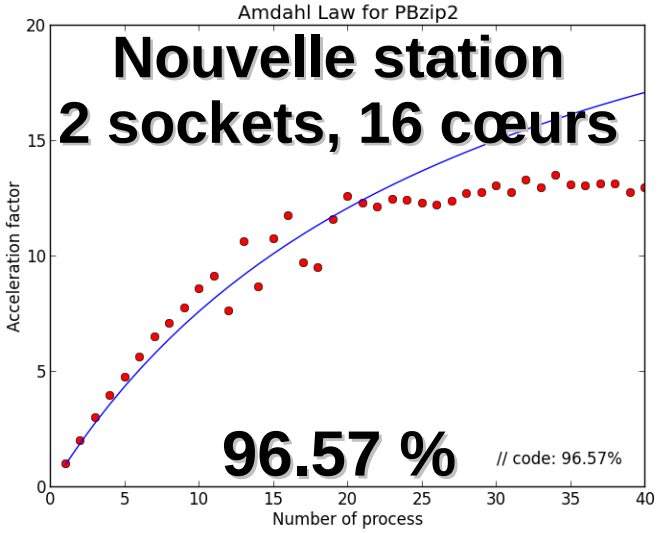
- Symptômes :
  - Sur un large nombre de cœurs, une efficacité de 70 % to 80 %
  - Grandes variations sur les machines double sockets
  - Décroissance de performances sur les charges élevées avec les vieux processeurs

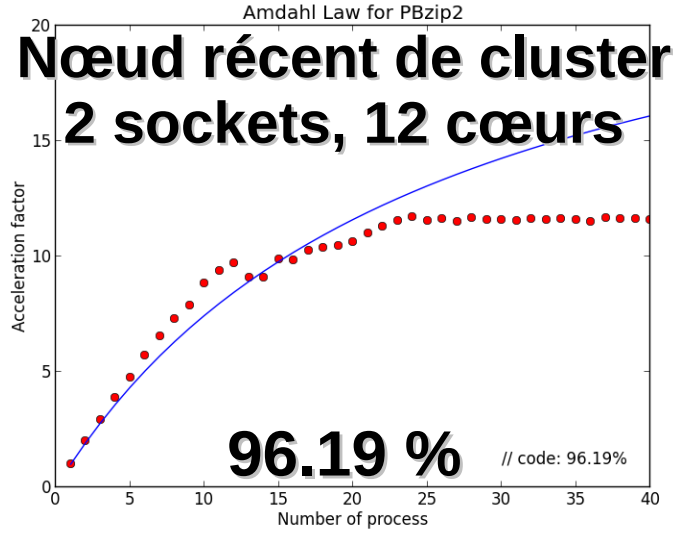


Loi d'Amdahl : « Fitting images » !

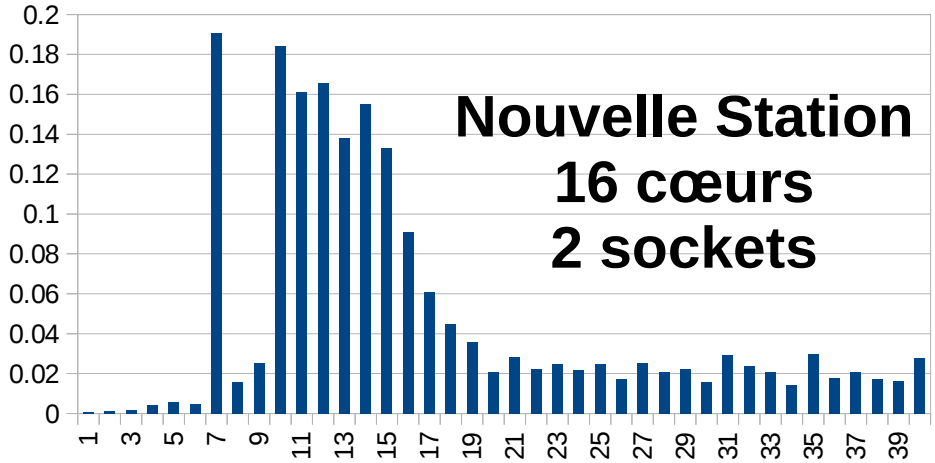


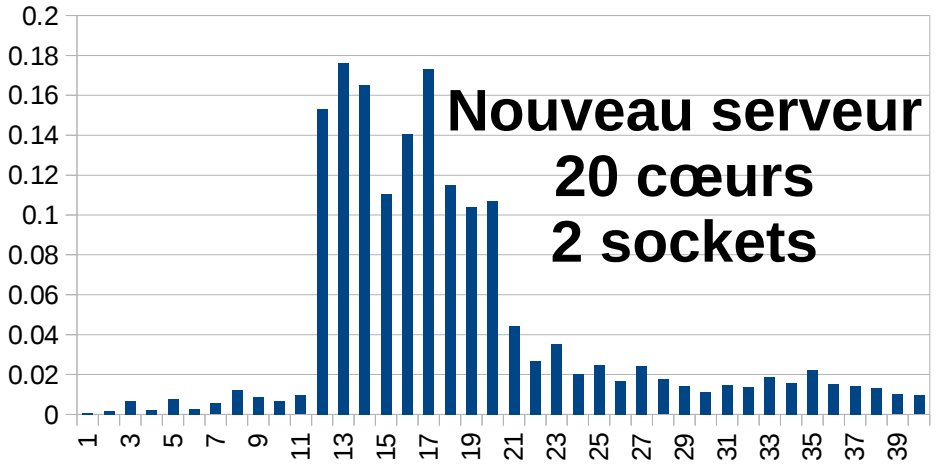




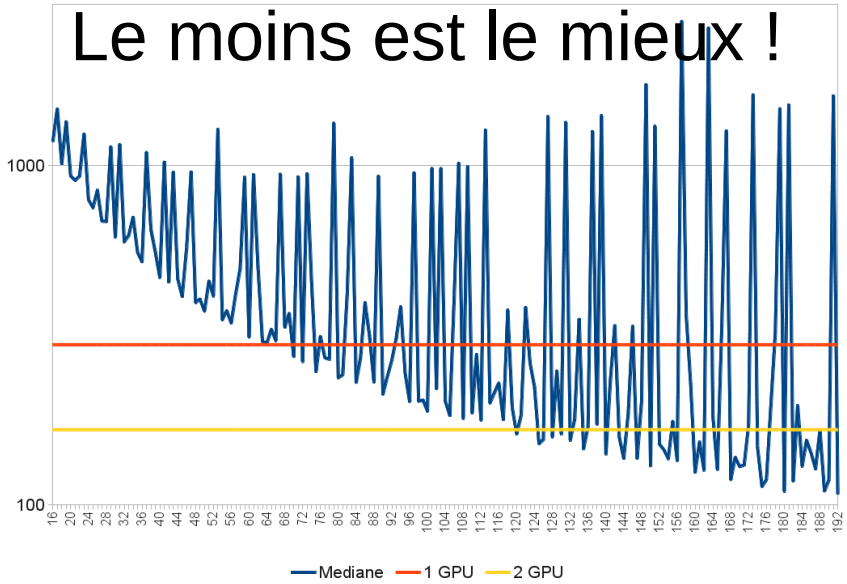


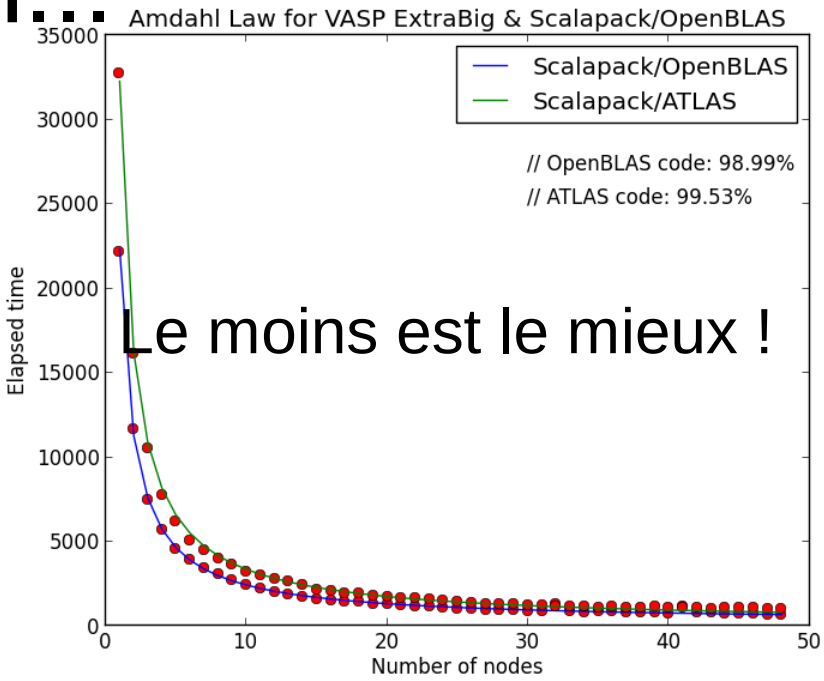
Et pire, la variabilité
Variabilité = Stddev/Median





Applications MPI, retour à la science
Loi d'Amdahl, en vrai...





Lammps, application de dynamique moléculaire

- De 16 à 192 NP
- Comparaison à des GPGPU
- 99.96 % //ized (record !)
- À l'époque, 2GPGPU équivaut 120 NP

Application DFT VASP

- De 1 à 48 NP
- 99 % à 99.5 % //ized
- OpenBLAS vs ATLAS...

Parallélisme sur CP2K
Du point de vue d'un utilisateur

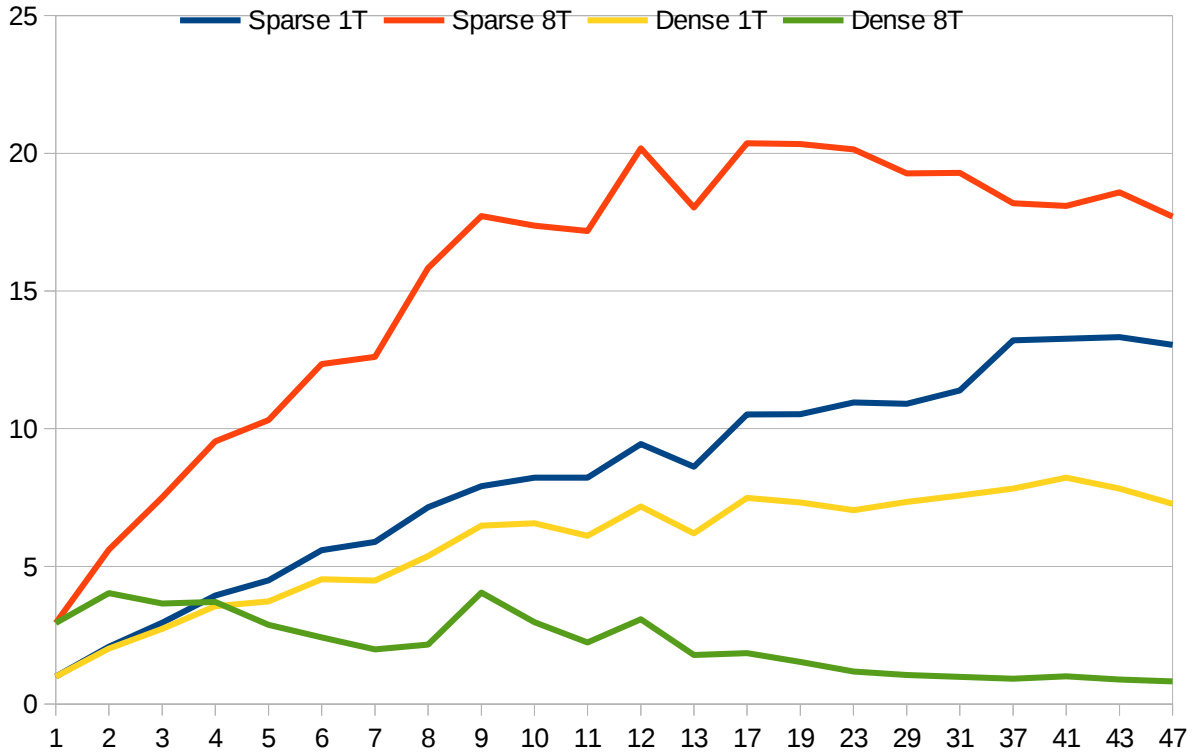
- 4 versions (en fait 8) :
  - Séquentiel : sopt
  - Parallélisé sur nœud (OpenMP & Pthreads) : ssmp
  - Parallélisé sur cluster (MPI) : popt
  - Parallélisé sur cluster & nœud : psmp
- 2 implémentations sur BLAS :
  - ATLAS : routines séquentielles BLAS
  - OpenBLAS : routines multithreadées BLAS
- Comment contrôler le parallélisme PR en Thread ou Processus ?
  - MPI : mpirun -np P <MyCode>
  - OpenMP & Pthreads : OMP_NUM_THREADS=T <MyCode>
  - Hybride : mpirun -np P -x OMP_NUM_THREADS=T <MyCode>

Parallélisme sur CP2K
Banc de Test

- Cluster : 48 nœuds R410, GE & Infiniband QDR
- Nœud : 2 sockets avec 4 cœurs chacun, 24 GB RAM
- OS : Debian Wheezy sur SIDUS, kernel 3.14
- Dépendances logicielles : standard & rétroportage
  - OpenMPI 1.4.5, FFTw3 3.3.2, OpenBLAS 0.2.8
  - Code : CP2K 2.5.1, version hybride : OpenBLAS & psmp
- Entrée : test H2O-128
- Exploration de 1 à 48 nœuds :
  - Mode Sparse & Dense (comment remplir les nœuds)
  - 1 Thread & 8 Threads (distribution sur les cœurs)

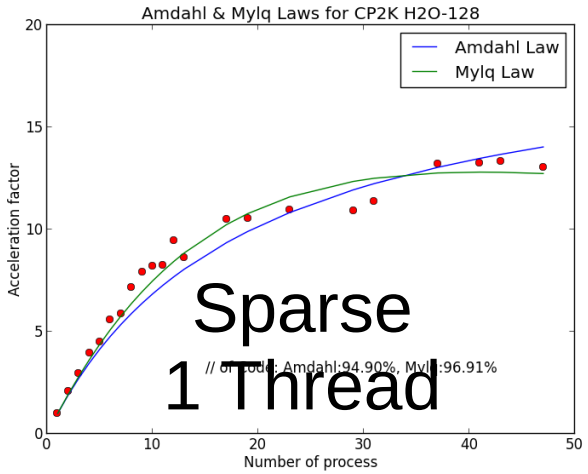
Parallélisme sur CP2K
Résultats (& Conclusions) !

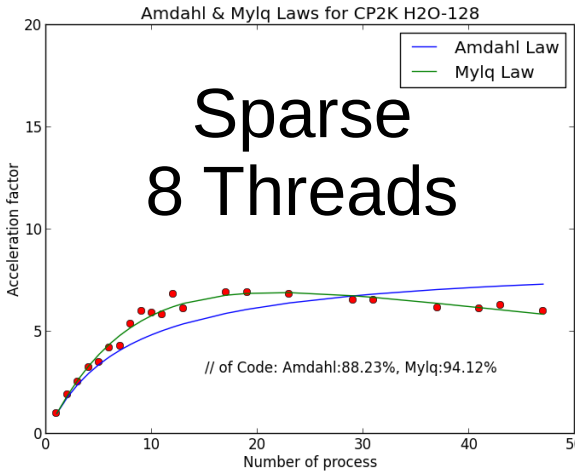
- Un découvreur de nombres premiers « bizarre » :
  - Only (booouuuh !) : 1 à 13, 17, 19, 23, 29, 31, 37, 41, 43, 47 (les autres plantent...)
- Scalabilités en vrac, et estimation du « p » d'Amdahl !
  - Distribué 1T : 95 %
  - Distribué 8T : 88 %
  - Dense 1T : 90 %
- Moralité :
  - L'hybride meilleur
  - Le distribué meilleur
  - Accélération jusqu'à 20
  - Loi d'Amdahl pas terrible

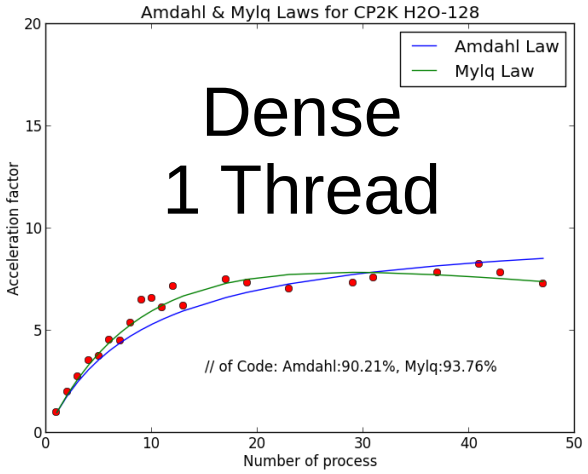


Parallélisme sur CP2K
Comment « fitter » au mieux ?

- Addition du facteur de correction dans la loi d'Amdahl :
  - de 1/(1-p+p/N) à 1/(1-p+p/N+c*n)
  - c : Constante de Mylq
- Correction linéaire finalement cohérente et efficace
- Moralité, corrigez la loi d'Amdahl en la loi de Mylq ;-)



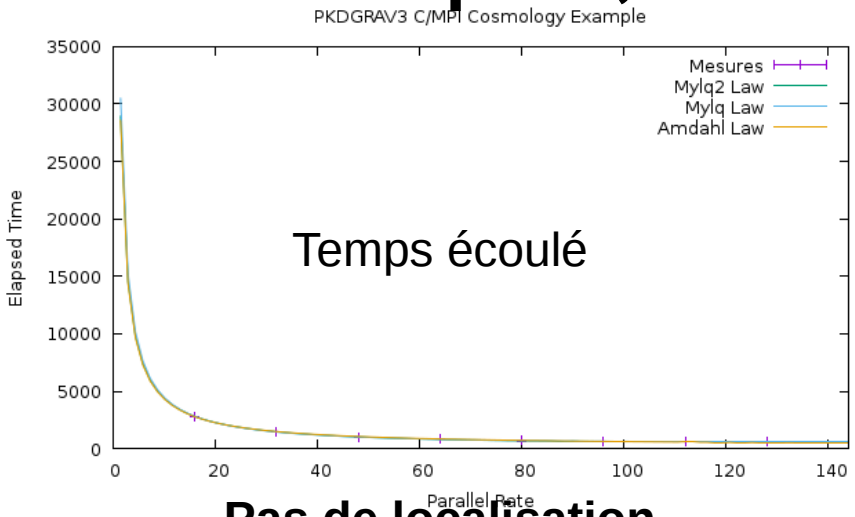


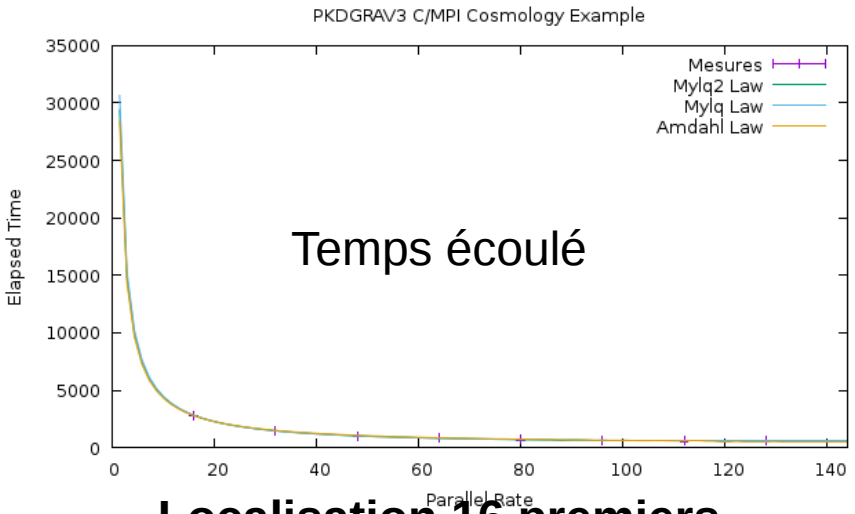


Et sur un code astrophysique ? PKDGRAV3

- Cluster : 8 nœuds C6320, GE & Infiniband FDR
- Nœud : 2 sockets avec 8 cœurs chacun, 128 GB RAM
- OS : Debian Stretch sur SIDUS, kernel 4.
- Dépendances logicielles :
  - OpenMPI, Pthreads, FFTW
- Entrée : test exemple cosmology.par, Grid de 128^3 à 256^3
- Exploration de 1 à 8 nœuds :
  - MPI : NP de 16 à 128, OMP_NUM_THREADS=1
  - MPI/Pthreads : NP de 1 à 8, OMP_NUM_THREADS=16
  - Sans et avec hwloc-bind -p pu:00-15

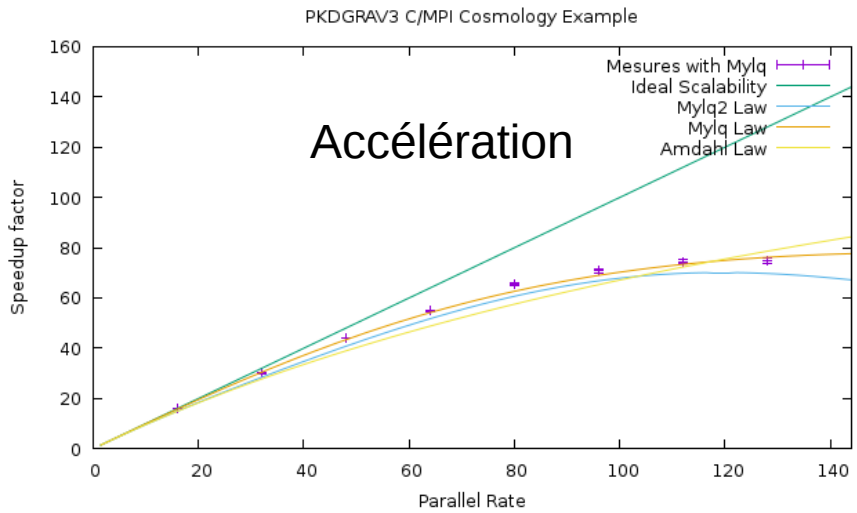
PKDGRAV3
MPI pur, de 16 à 128 de NP

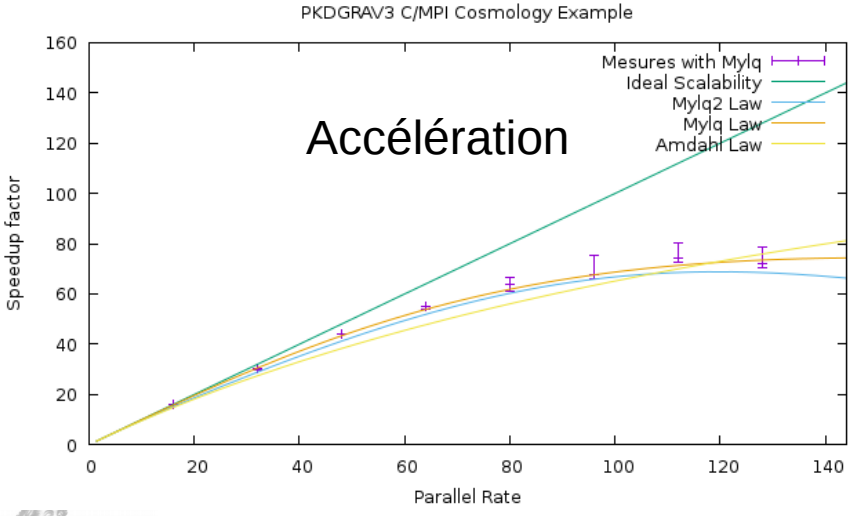




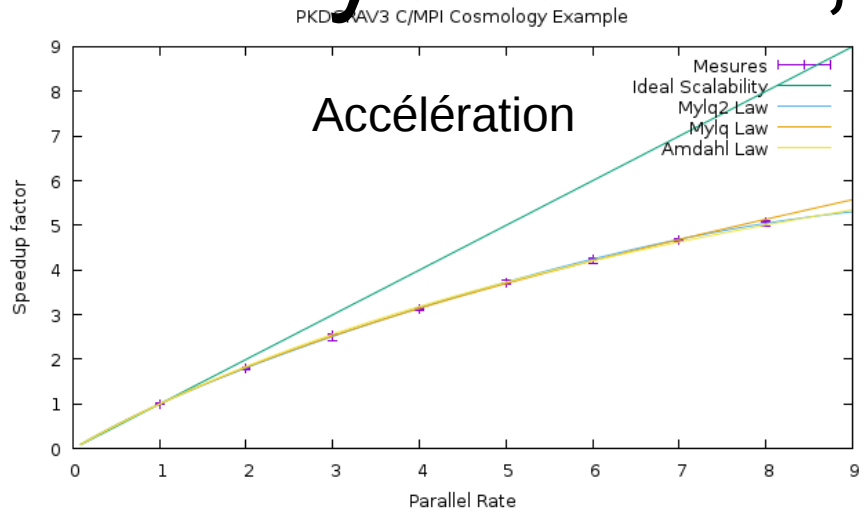
Pas de localisation

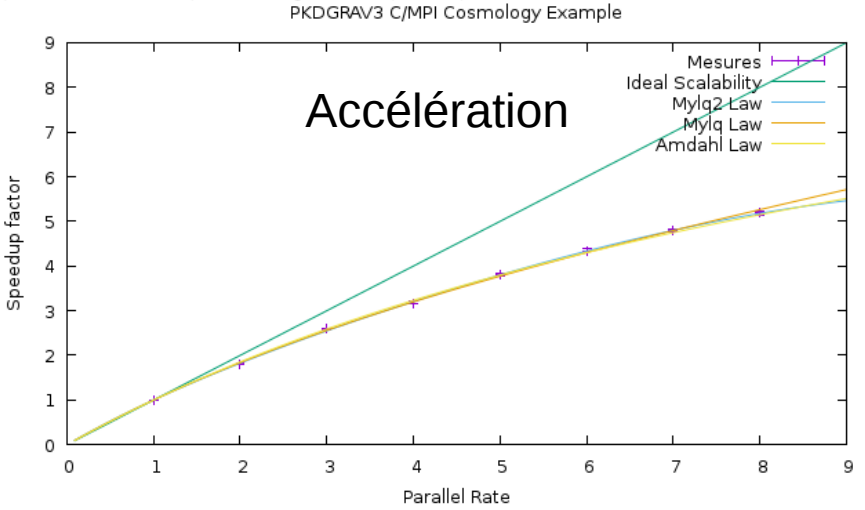
Localisation 16 premiers



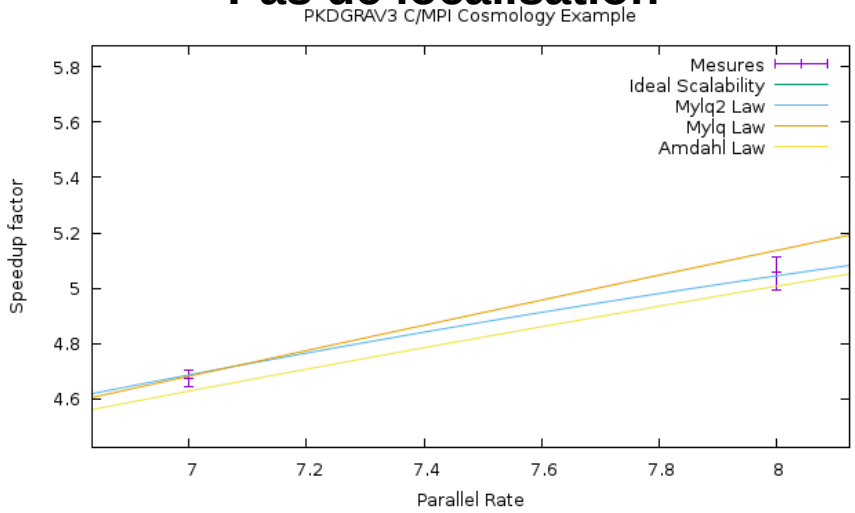


PKDGRAV3
Hybrid MPI, de 1 à 8 de NP

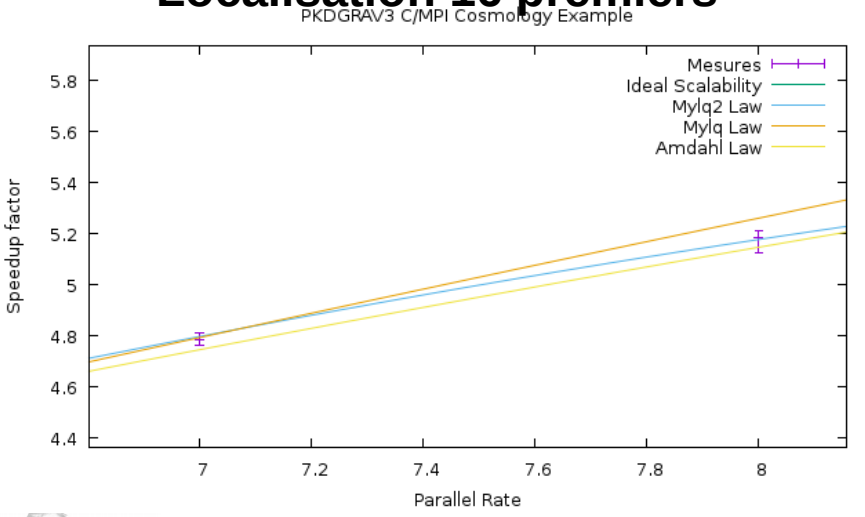




Pas de localisation



Localisation 16 premiers

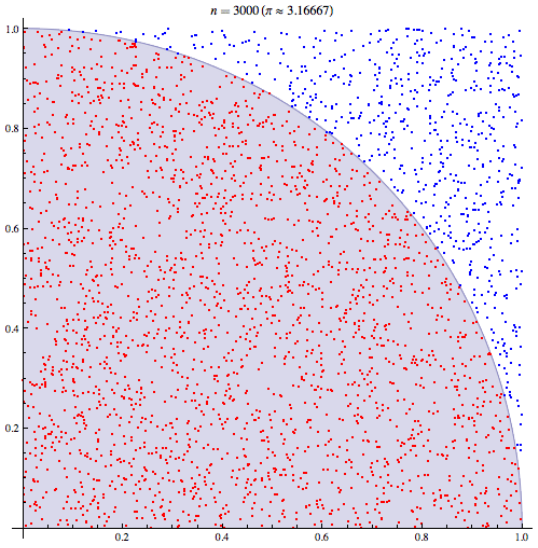


PKDGRAV3 : conclusions

- En MPI pur, moins de variabilité sur le délocalisé
- En hybride, moins de variabilité sur le localisé
- La loi de Mylq est bien pour du MPI pur
- La loi de Mylq2 est meilleur sur l'Hybride
- Moralité :
  - La loi de Mylq du premier ordre est efficace pour des codes MPI
  - La loi de Mylq du second ordre est meilleur pour les codes hybrides
  - L'influence de la « localisation » est à étudier soigneusement...

Et pour des implémentations simples ?
PiMC : Pi by Dart Board Method

- Exemple classique du calcul de Monte Carlo
- Implémentation parallèle : distribution
  - De 2 à 4 paramètres
    - Nombre total d'itérations
    - Régime de Parallélisme (PR)
    - (Type de variable : INT32, INT64, FP32, FP64)
    - (RNG : MWC, CONG, SHR3, KISS)
  - 2 observables simples :
    - Estimation de Pi estimation (just indicative, Pi n'est pas rationnel :-))
    - Temps écoulé

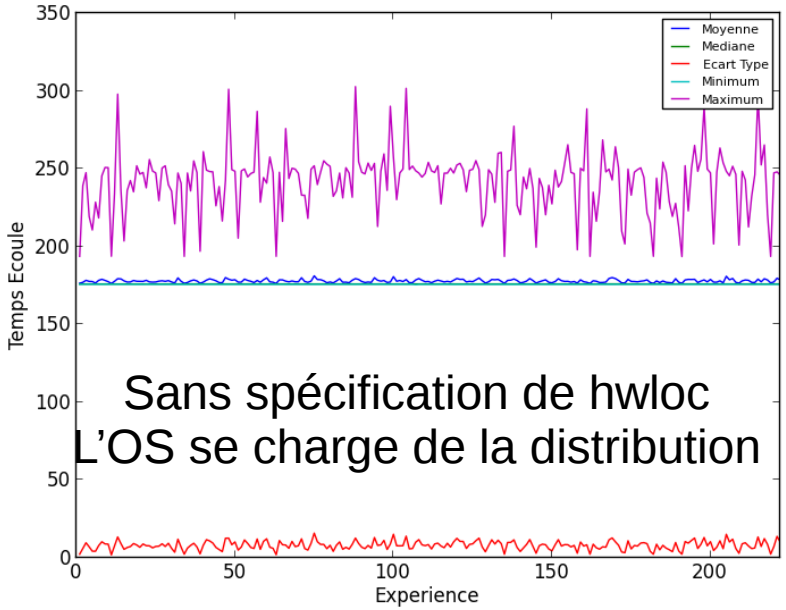


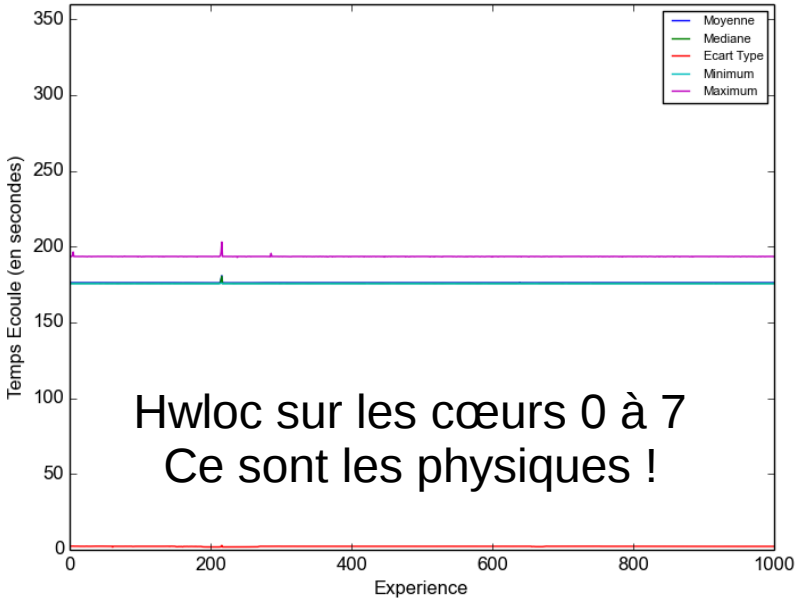
PiMC juste pour la 6e école des Houches

- Banc d'essai :
  - Matériel : 64 R410 avec 8 cœurs en mode HT,
    - Interconnexion Infiniband QDR (40 Gb/s)
  - OS : Debian Jessie SIDUS
  - Logiciel : OpenMPI/C
- Expériences :
  - Communications réduites au minimum
  - 1 Piterations : 10^{12} également distribuées
  - Régime de parallélisme de 1 à 512 (distribution par saut)
  - 40 lancements pour chaque régime
  - La métrologie est assurée par le programme « time »
  - /usr/bin/time mpirun.openmpi -np $PR -mca btl self,openib,sm -hostfile $MyHostFile -loadbalance hwloc-bind -p pu:$AFF /scratch/root/bench4gpu/Pi/C/MPI/Pi_MPI_FP32_MWC $ITERATIONS

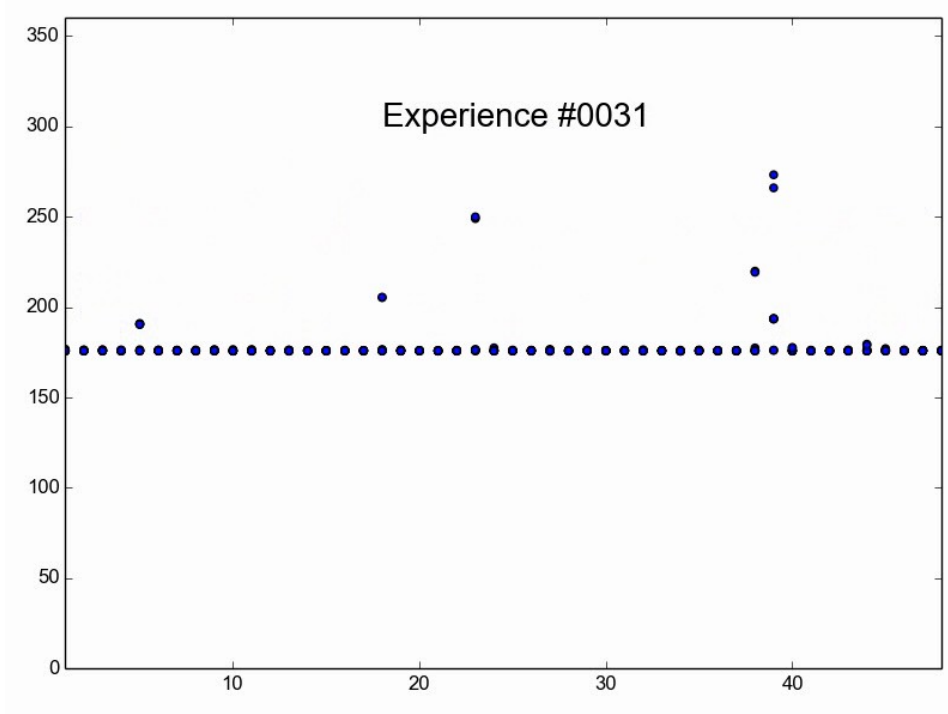
PiMC : pourquoi une « affinité » ?

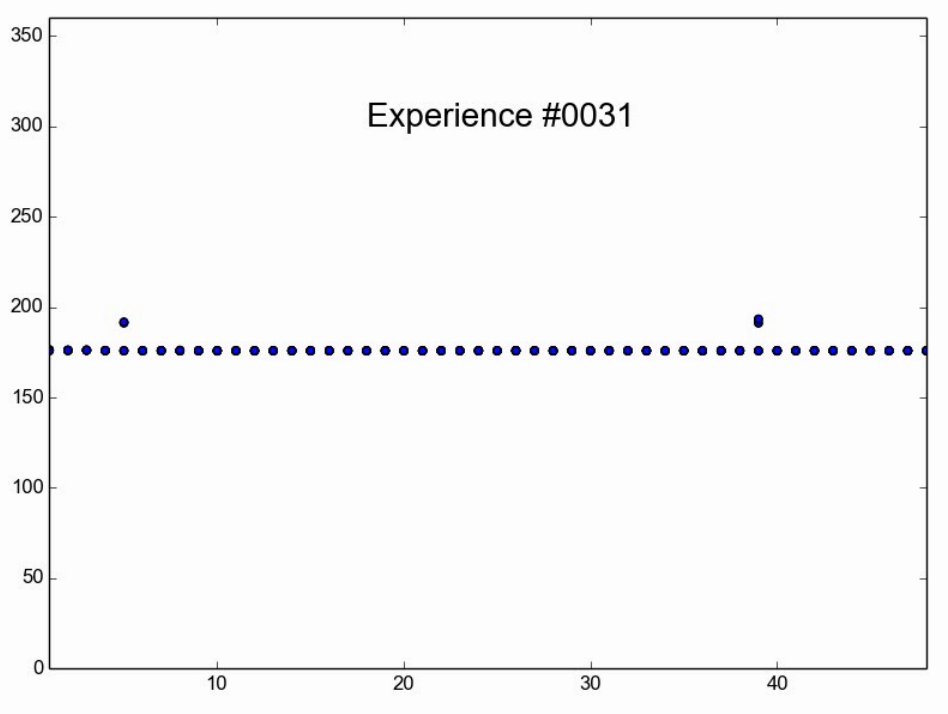
- Durant la qualification d'un cluster de 48 nœuds
- Des centaines de lancements pour éprouver l'infra...
- Moralité : « localiser » les processus est pas mal...





PiMC : pourquoi une « affinité » ? Temps écoulé sur 48 nœuds





Pas d'affinité...

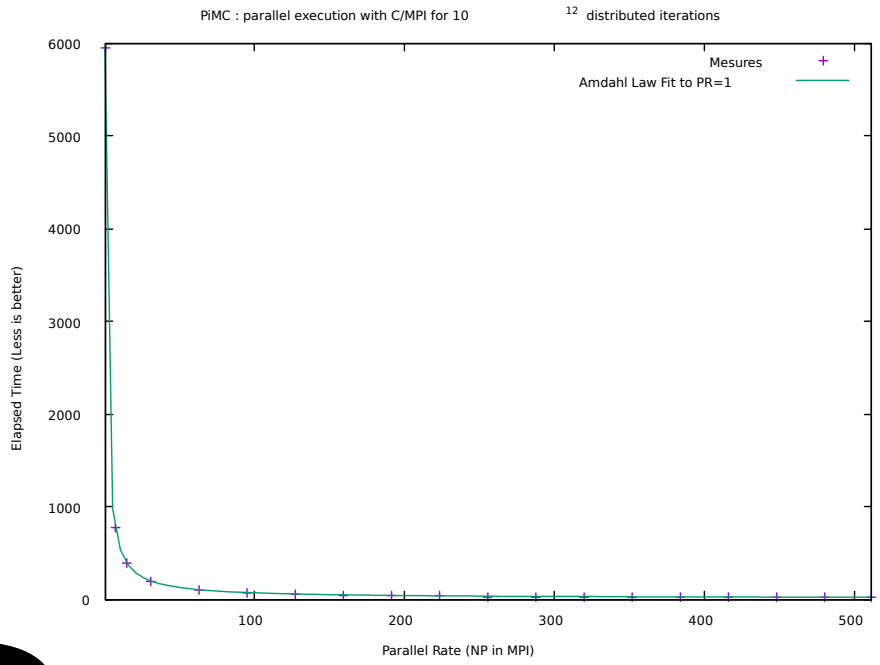
Affinité sur les 8 premiers

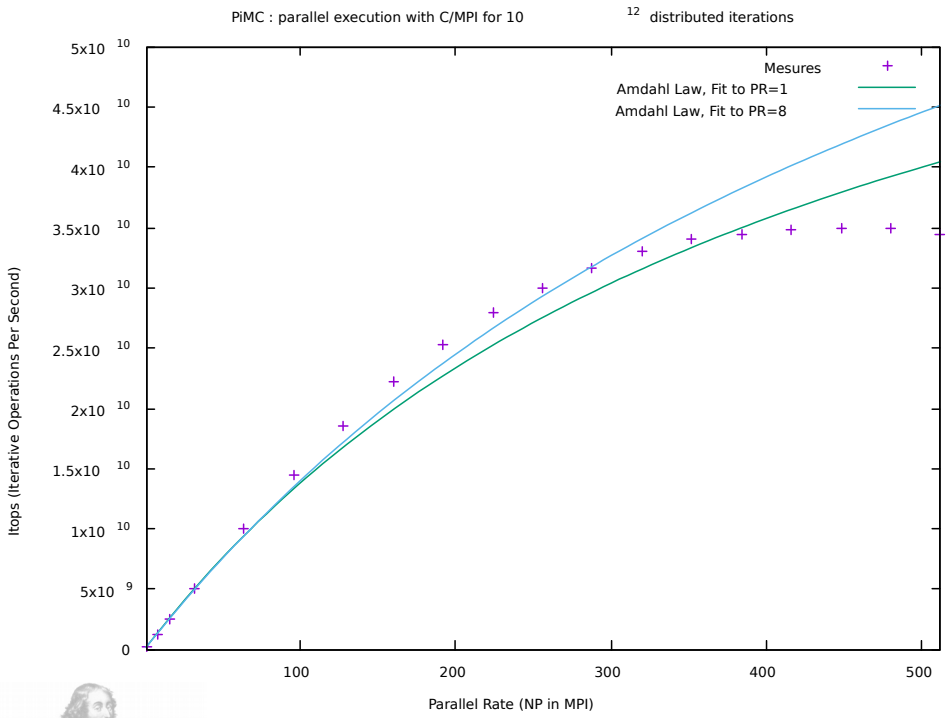
PiMC : et les résultats sont...

NP	Itops	Speedup 1	Speedup 8	Total Time	Variability %	Average	Median	Stdev	Minimum	Maximum
1	1.68E+08	1.00	1.05	5952	0.04	5953.22	5952.11	2.46	5950.32	5958.86
8	1.28E+09	7.62	8.00	6245	0.05	780.54	780.65	0.35	780.02	781.05
16	2.55E+09	15.21	15.96	6263	0.04	391.41	391.41	0.16	391.04	391.71
32	5.08E+09	30.22	31.71	6302	0.07	196.95	196.93	0.13	196.82	197.53
64	9.96E+09	59.30	62.22	6424	0.08	100.39	100.38	0.08	100.25	100.63
96	1.45E+10	86.31	90.56	6621	0.25	68.94	68.97	0.18	68.32	69.53
128	1.86E+10	110.86	116.32	6872	0.55	53.75	53.69	0.29	53.14	54.74
160	2.22E+10	132.12	138.63	7208	0.75	45.09	45.05	0.34	44.47	46.28
192	2.53E+10	150.53	157.95	7592	0.59	39.52	39.54	0.23	38.85	40.20
224	2.80E+10	166.38	174.57	8014	0.81	35.80	35.78	0.29	35.21	37.10
256	3.00E+10	178.80	187.60	8522	0.74	33.32	33.29	0.25	32.78	34.28
288	3.17E+10	188.54	197.82	9092	0.82	31.58	31.57	0.26	31.04	32.30
320	3.30E+10	196.28	205.94	9704	1.04	30.37	30.33	0.31	29.87	31.26
352	3.40E+10	202.14	212.10	10365	1.43	29.52	29.45	0.42	28.83	30.58
384	3.44E+10	204.86	214.94	11157	1.29	29.08	29.06	0.38	28.34	30.19
416	3.48E+10	207.14	217.34	11954	1.03	28.70	28.74	0.30	28.19	29.67
448	3.50E+10	208.08	218.32	12815	1.16	28.67	28.61	0.33	28.04	29.69
480	3.49E+10	207.97	218.21	13738	1.34	28.65	28.62	0.38	27.99	29.77
512	3.45E+10	205.10	215.20	14858	1.28	29.10	29.02	0.37	28.58	30.18

PiMC : graphiquement Amdahl est-elle une bonne loi ?

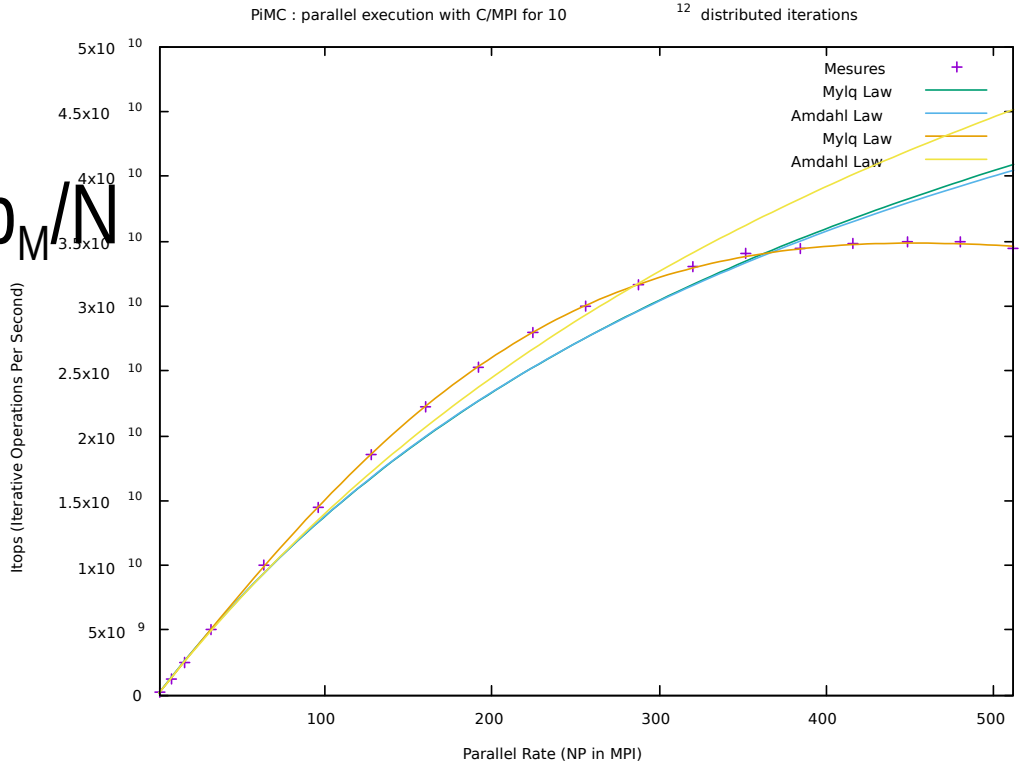
- Temps écoulé en secondes
  - Ca me semble correct…
- Performance en Itops
  - Fit to 1 : p=99.78 %
  - Fit to 8 : p=99.83 %





Evolution de la loi d'Amdahl en Mylq
Intégration d'une influence linéaire

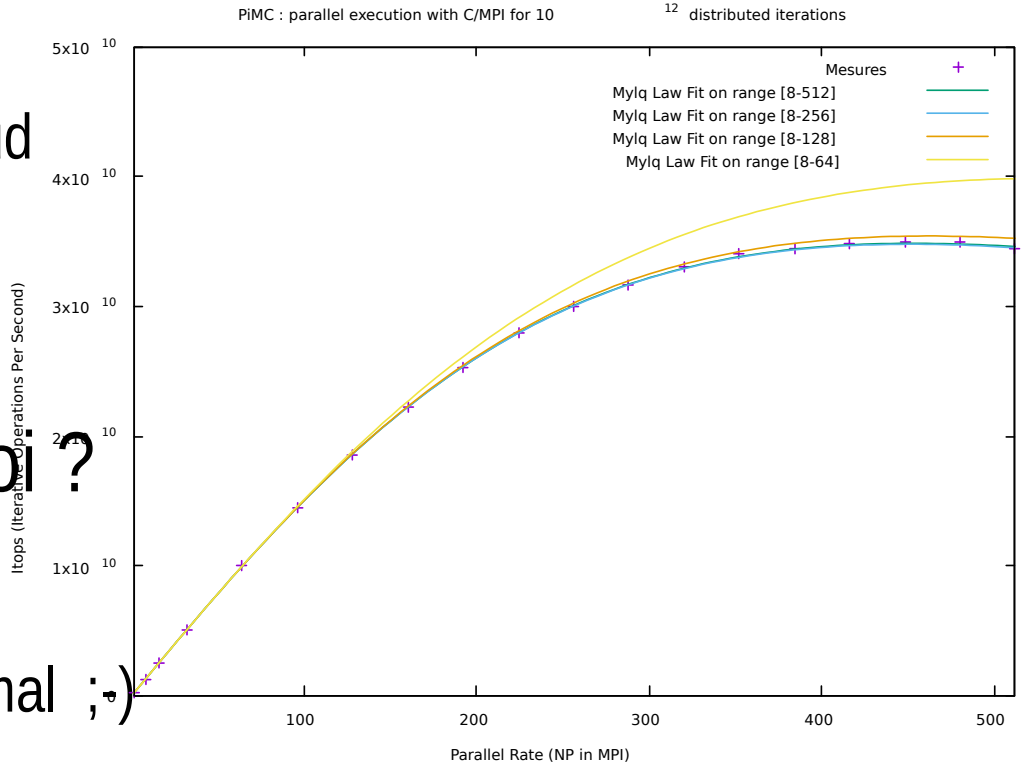
- Loi d'Amdahl : $T=s_A+p_A/N$
- Loi de Mylq : $T=s_M+c_MN+p_M/N$
- Signification de c_M :
  - Communications
  - Processus d'initialisation
  - and c_M~0.03,



- Et $p_{M\ normalisé}$ ~ 0.9998 si on exclut la valeur pour PR=1

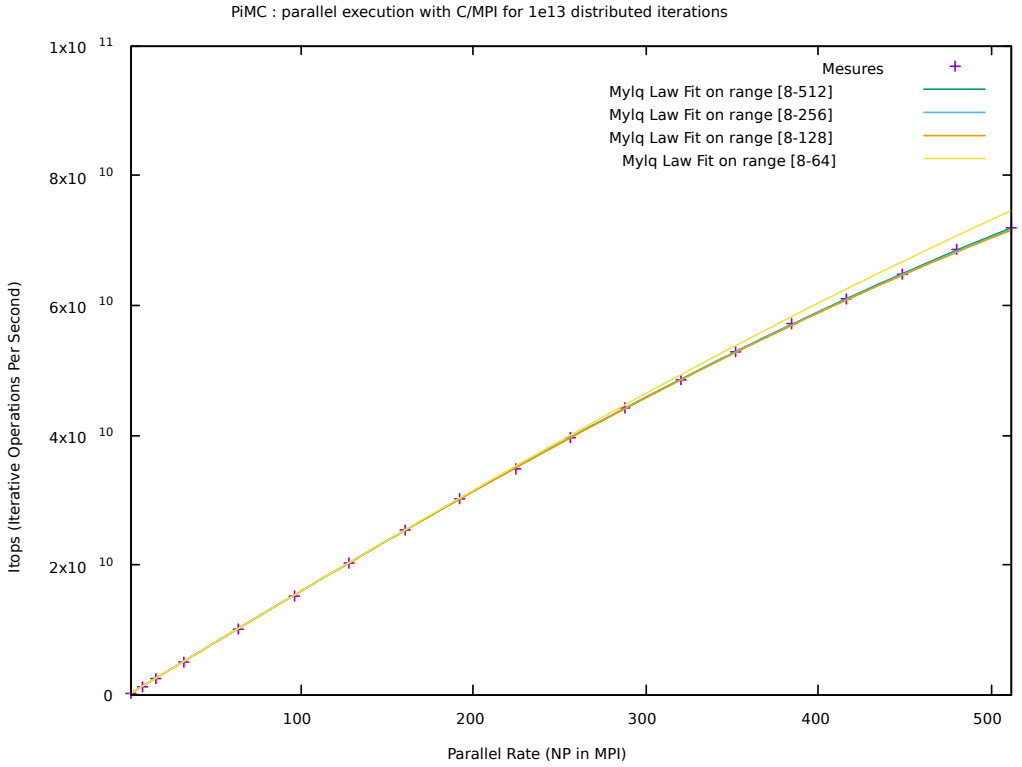
Loi de Mylq : Pourquoi exclure PR=1 ? Est-ce une loi plus prédictible ?

- Pourquoi exclure PR=1
  - Mécanismes internes au nœud
  - Effets de l'OS
  - Effets du Processeur : Turbo
- Plus prédictible comme loi ?
  - Essai de « fit » : 1/2,1/4,1/8
  - Pour 1/4, ça ne marche pas mal ;-)
- Mais d'autres effets à intégrer...



Influence sur le temps écoulé
Et si j'augmente le nb d'itérations ?

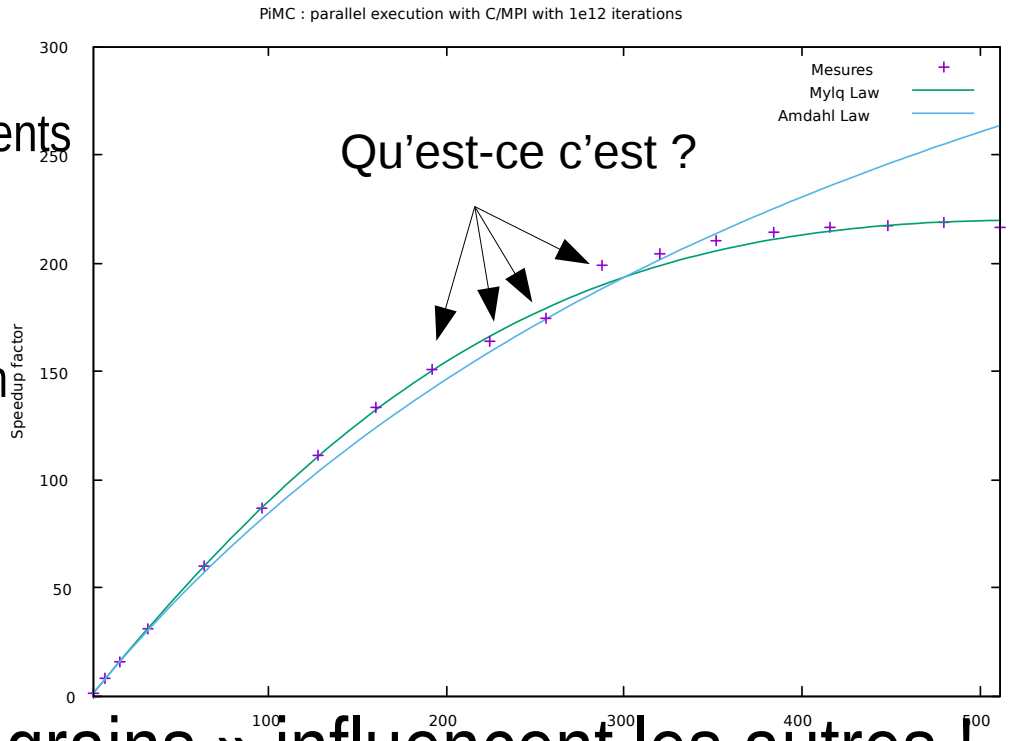
- De 10^{12} à 10^{13} itérations
- Accélération de 208 à 448
- Efficacité de 40 % à 87 %
- Itops de 34 à 72 Gitops
- Paramètres de Mylq :
  - p_M déduit de 0.99998
  - c_M=0.032 (le précédent 0.03)



- Moralité : ne soyez pas trop « léger » sur vos sets d'entrée...

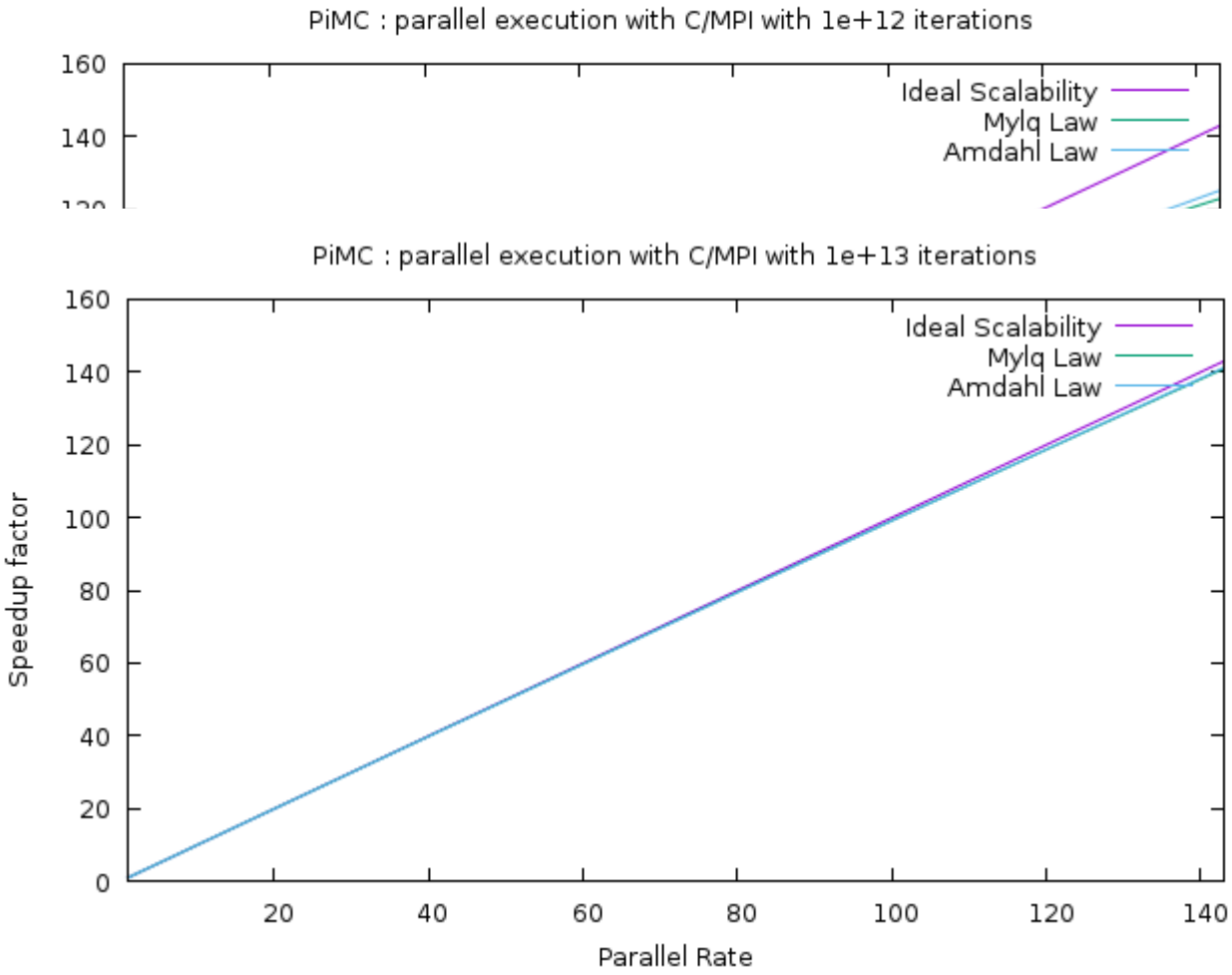
Pourquoi ce qui est précédent n'est pas très honnête...

- Naguères, Mylq n'était pas si bon, pourquoi ?
  - Défaut de statistiques ?
    - Pour chaque PR, 10 lancements
    - Distribution sur 64 nœuds
      - Jobs concurrents
      - Exclusif dans l'exécution
    - Variabilité autour 1 %
  - Solution :
    - Lancements exclusifs



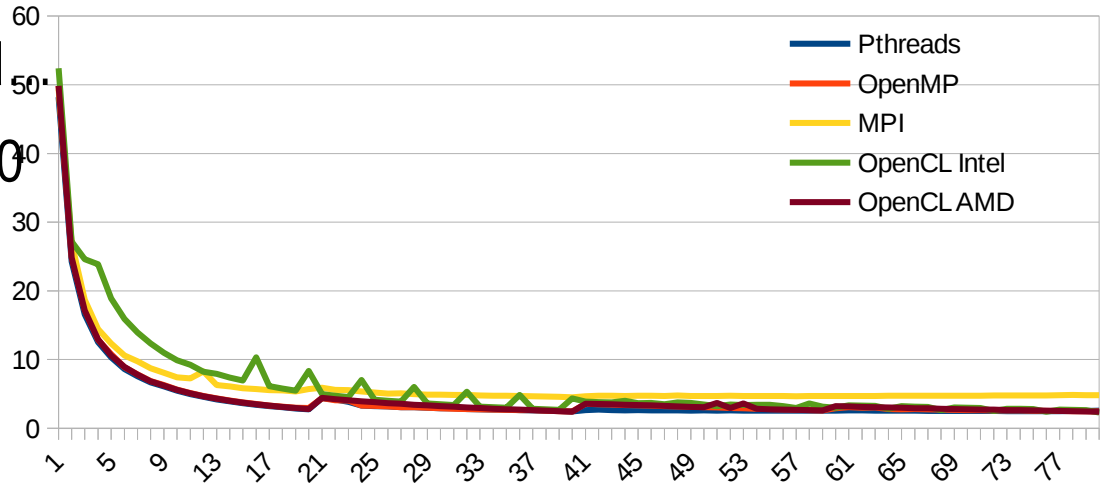
- Même les codes « gros grains » influencent les autres !

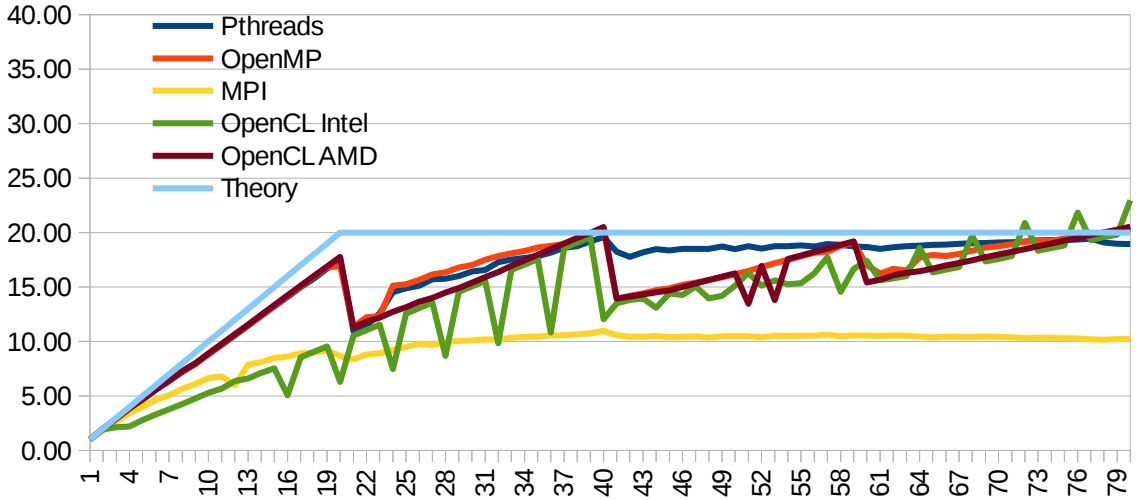
Et sur un cluster récent ? Les mêmes comportements ?



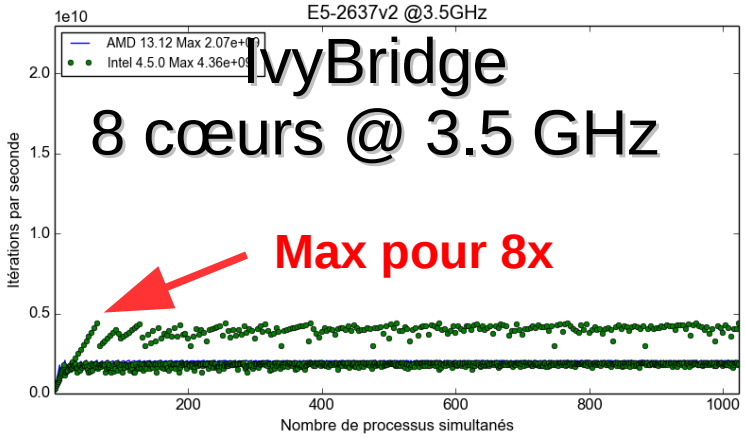
Et les autres parallélismes ? C'est pire ! Petit exemple

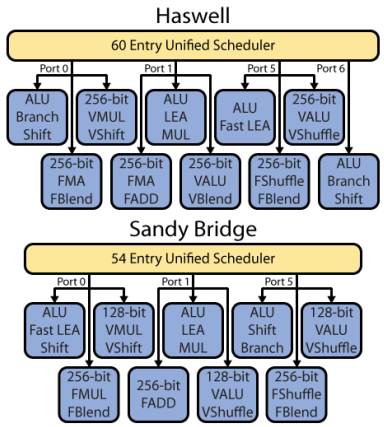
- Retournons à l'intérieur du nœud
- Un Dell serveur PowerEdge R620
  - Bi-socket, 10-cœurs : 20 cœurs
  - Mode Hyperthreading activé : 40
- Implémentations parallèles
  - MPI en C
  - OpenMP en C
  - Pthreads en C
  - OpenCL en Python
    - OpenCL par AMD
    - OpenCL par Intel
- Finalement, pas mal OpenCL !

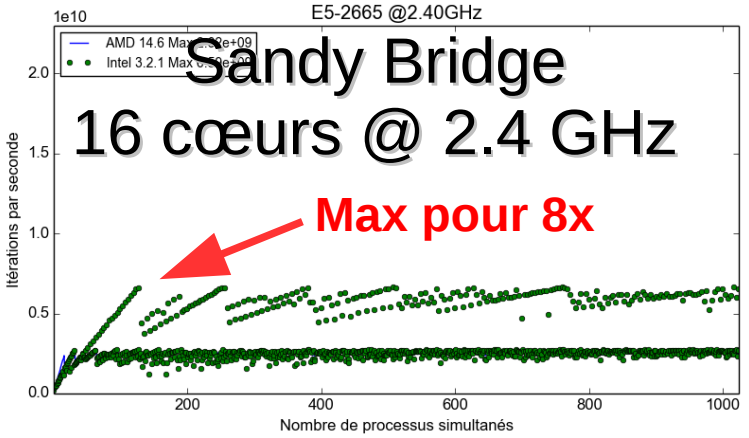


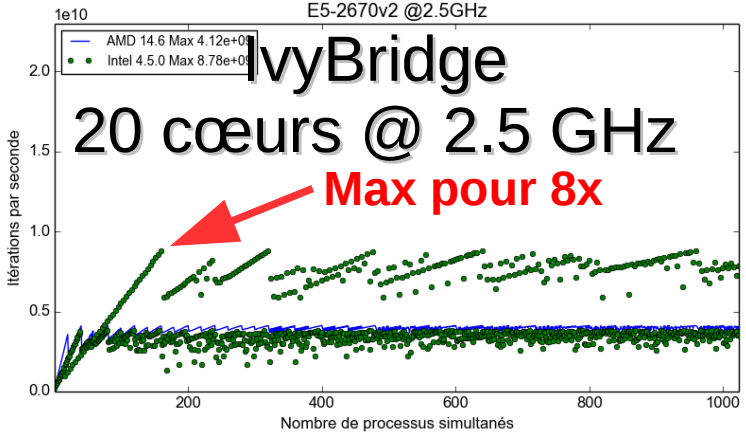


Et pour les PR >> Nombre de cœurs
EPU depend de l'architecture !



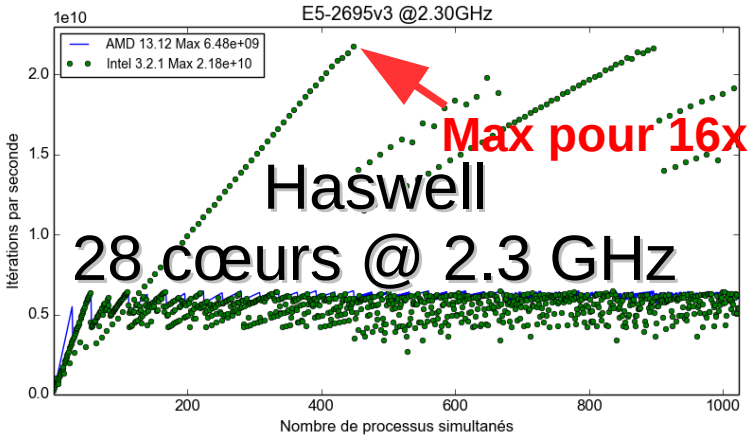






Intel x2,3 vs AMD
Period of 4
Max Perf :

- x8 Sandybridge
- x8 IvyBridge
- x16 Haswell



Position du GPU face aux autres? Les autres « accélérateurs »

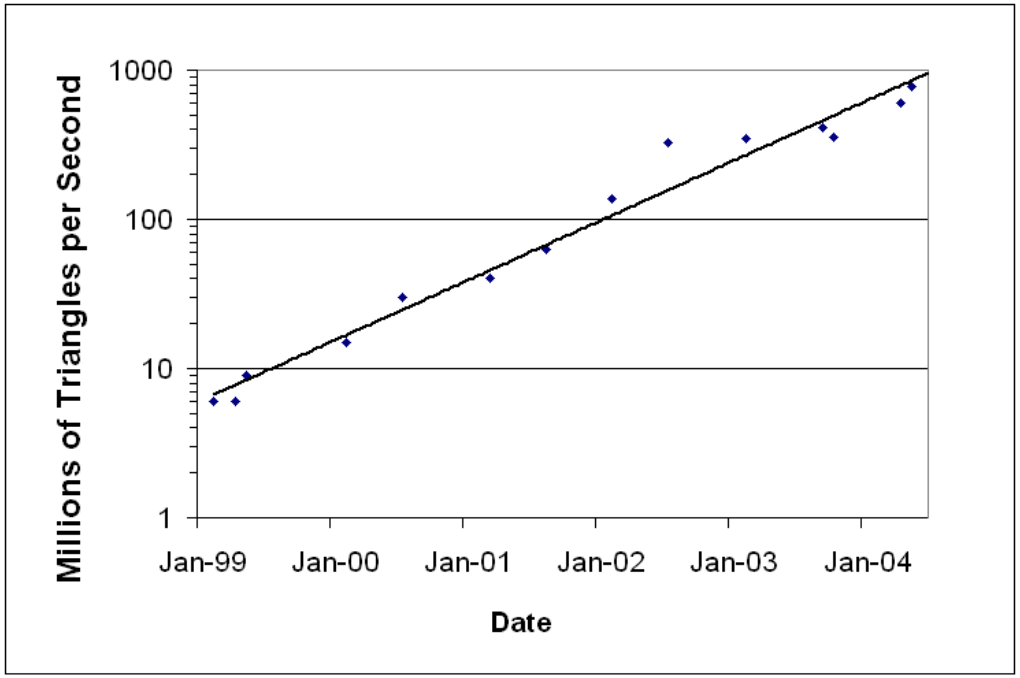
- Accélérateur ou la vieille histoire des coprocesseurs...
  - 1980 : 8087 (sur 8086/8088) pour les opérations en virgule flottante
  - 1989: 80387 (sur 80386) et son respect du IEEE 754
  - 1990 : 80486DX et l'intégration du FPU dans le CPU
  - 1997 : K6-3DNow ! & Pentium MMX : SIMD dans le CPU
  -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

« Au commencement de toute chose »

- « Nvidia Launches Tesla Personal Supercomputer »
- Quand : on 19/11/2008 Qui : Nvidia
- Où : sur Tom's Hardware
- Quoi : une carte PCIe C1060 PCIe avec 240 cores
- Combien : 933 Gflops SP (but 78 Gflops DP)

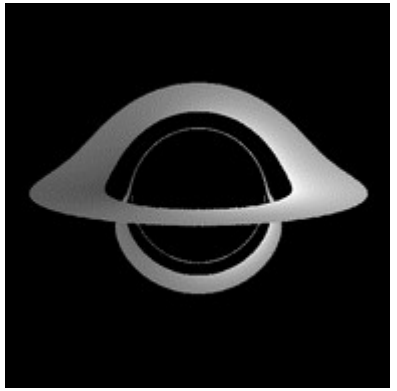
Pourquoi le GPU est « disruptif » ? Le « grand détournement » en HPC

- Dans une conférence à l'ENS-Cachan en février 2006, ceci...
  - x100 en 5 ans



- Entre 2000 et 2015
  - GeForce 2 Go/GTX 980Ti : de 286 à 2816000 MOpérations/s : x10000
- Pour un CPU « classique » : x100

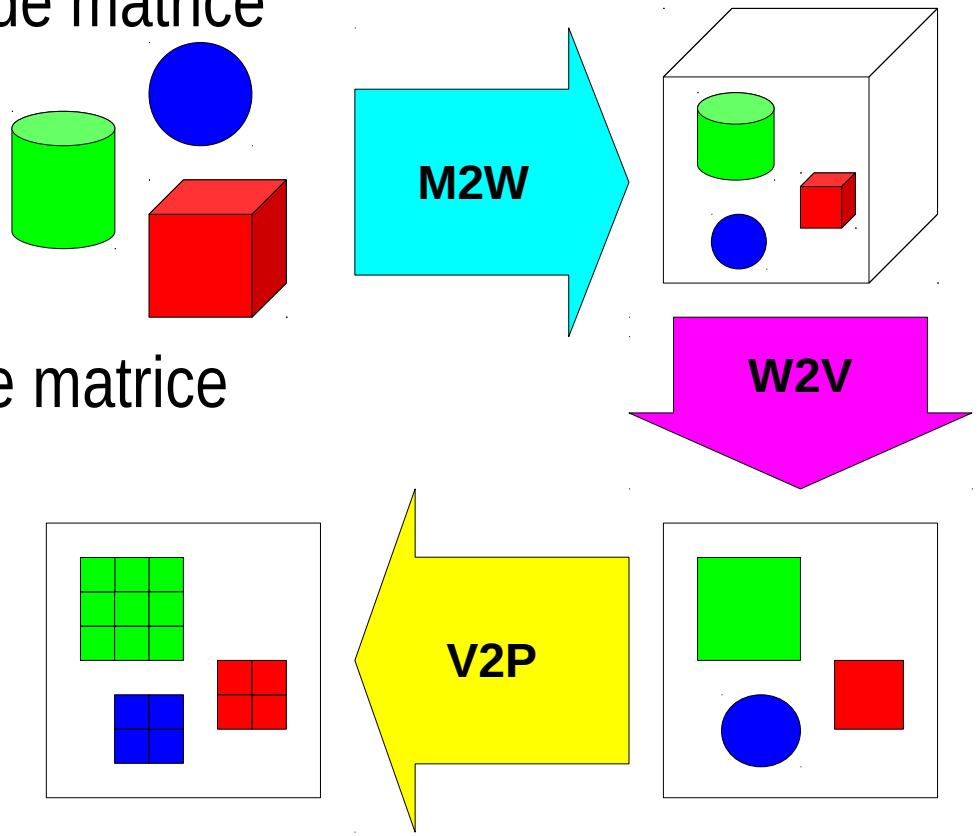
Pourquoi le GPU est-il si puissant ? Pour construire une image 3D !



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

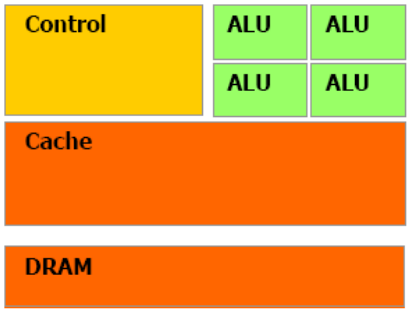
Pourquoi le GPU est-il si puissant ? Shadering & Calcul matriciel

- **Model 2 World** : 3 produits de matrice
  - Rotation
  - Translation
  - Mise à l'échelle
- **World 2 View** : 2 produits de matrice
  - Positionnement de la caméra
  - Direction de l'endroit pointé
- **View 2 Projection**
  - Pixellisation

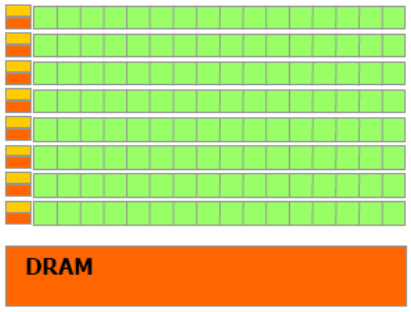


Donc, un GPU : c'est un « gros » Multiplicateur de Matrice

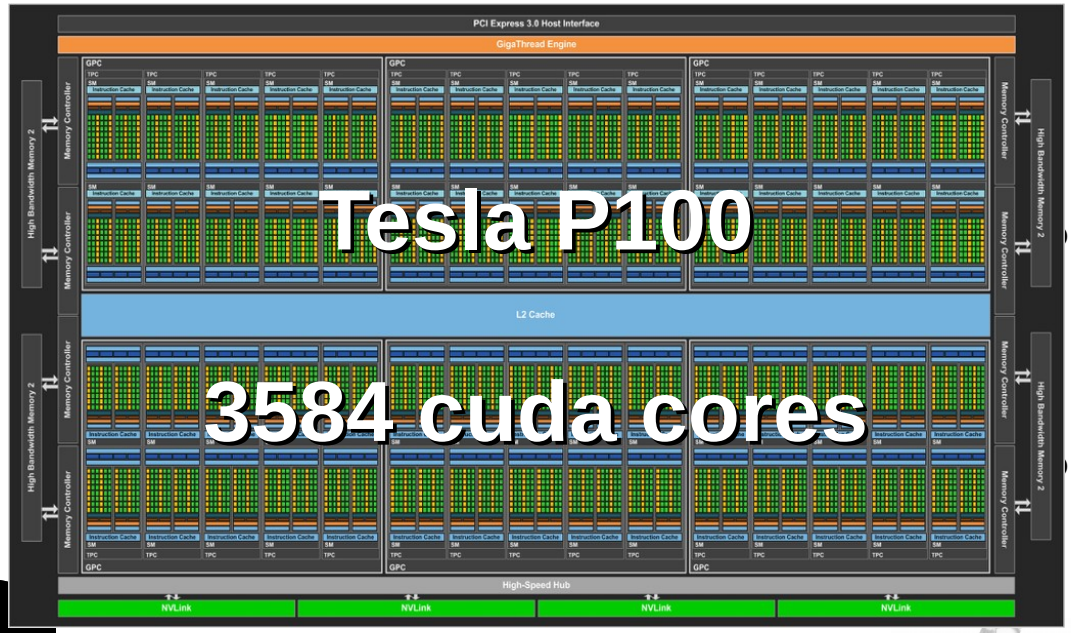
Combien de composants à l'intérieur ? Quelle différence entre GPU & CPU



CPU



GPU



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

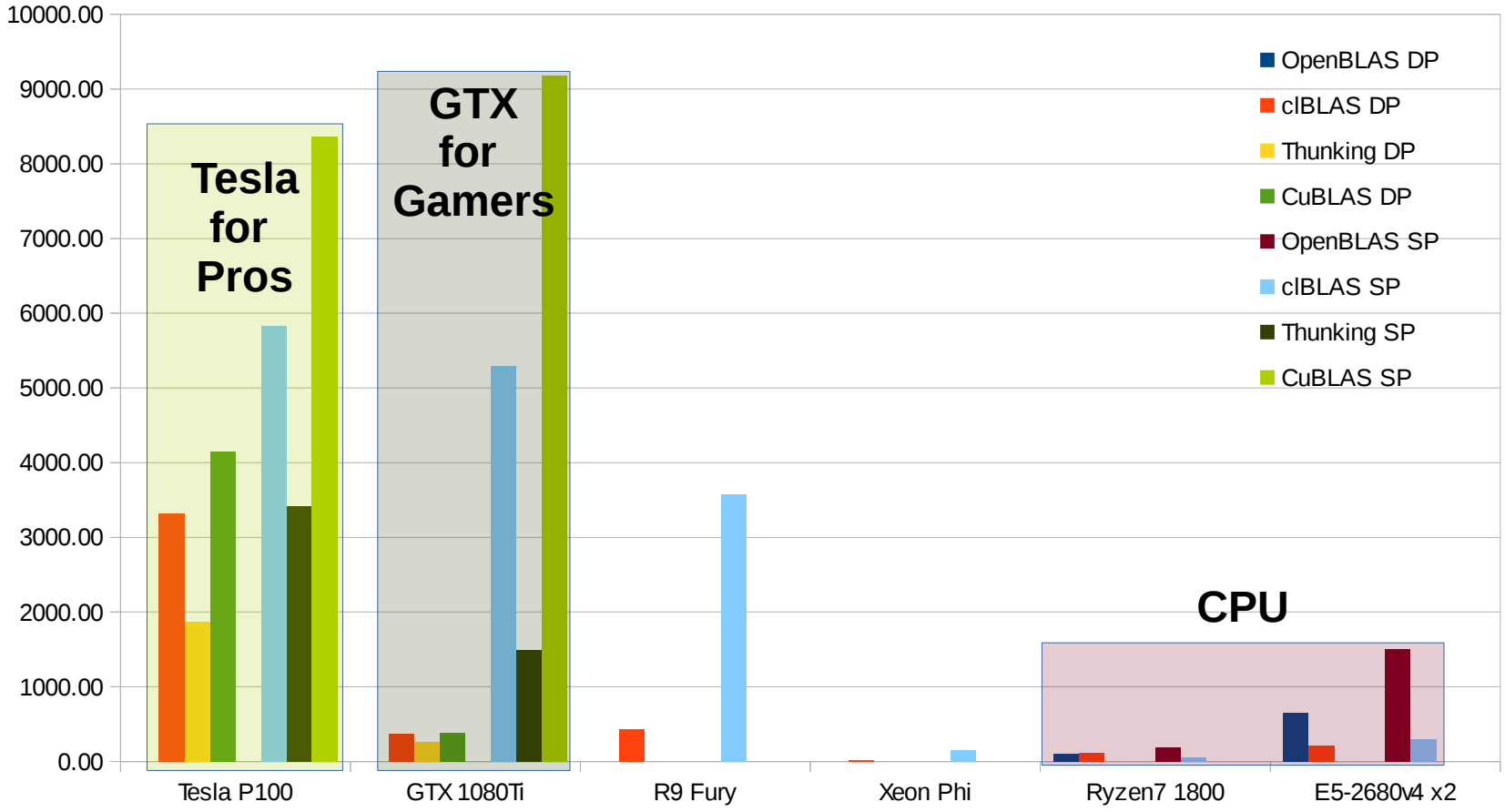
Programmation : 1993

- OpenGL, Glide, Direct3D, ...

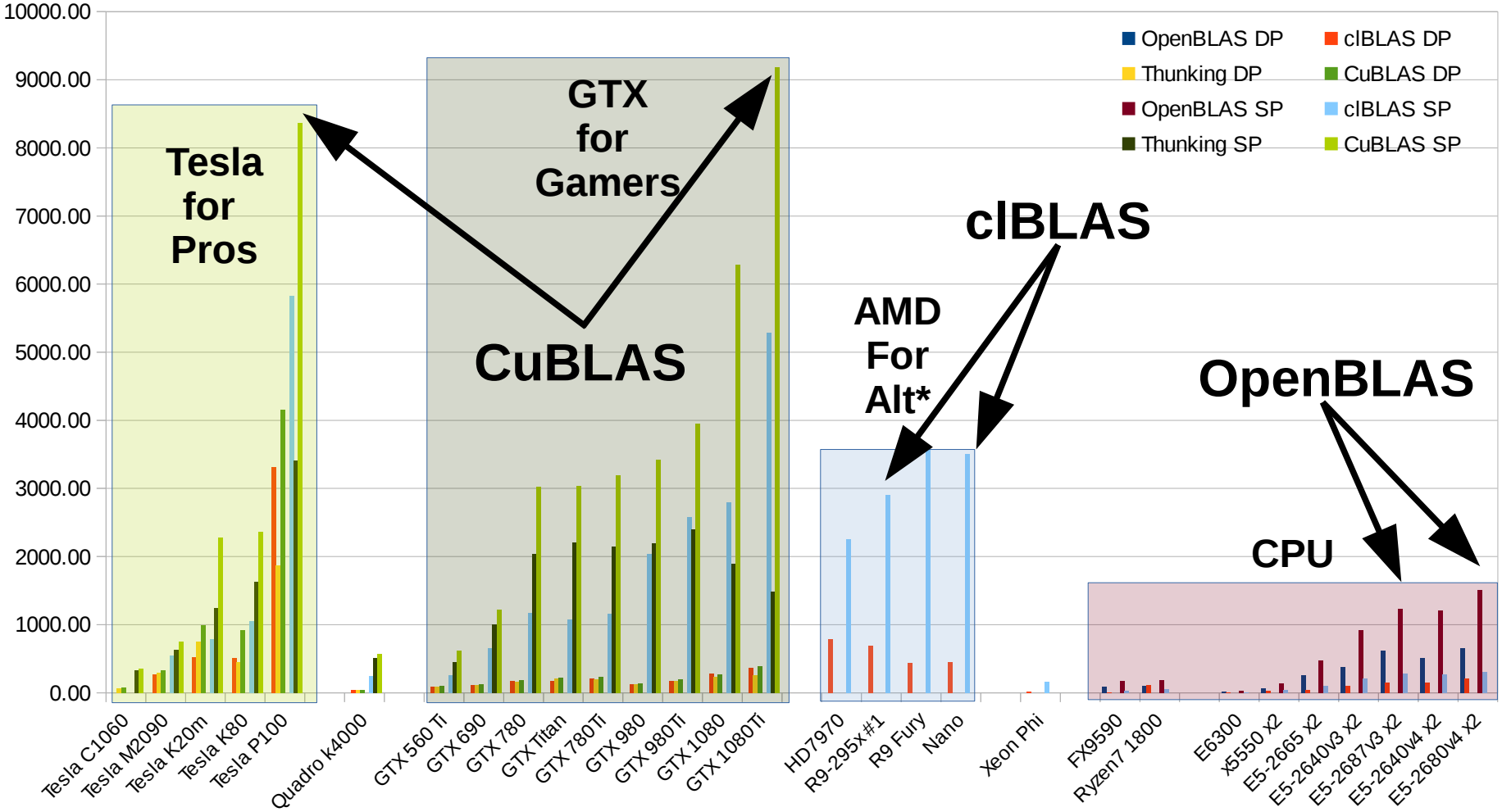
Généricité : 2002

- CgToolkit, CUDA, OpenCL

Est-ce que le GPU est si puissant ? Pour mes meilleures MyriALUs...



BLAS, le match : CPU vs GPU
xGEMM quand x=D(P) or S(P)

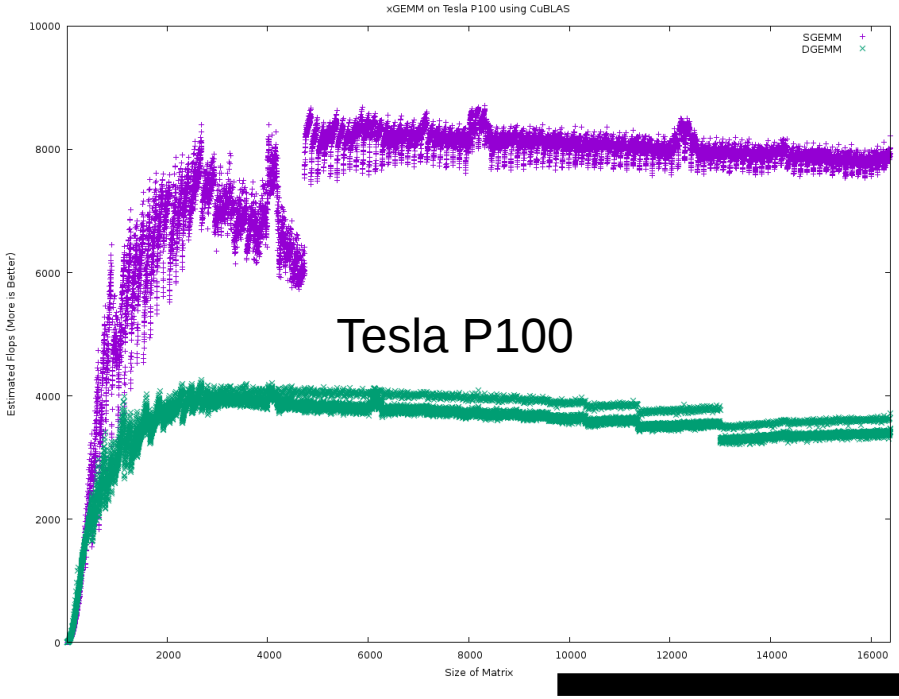


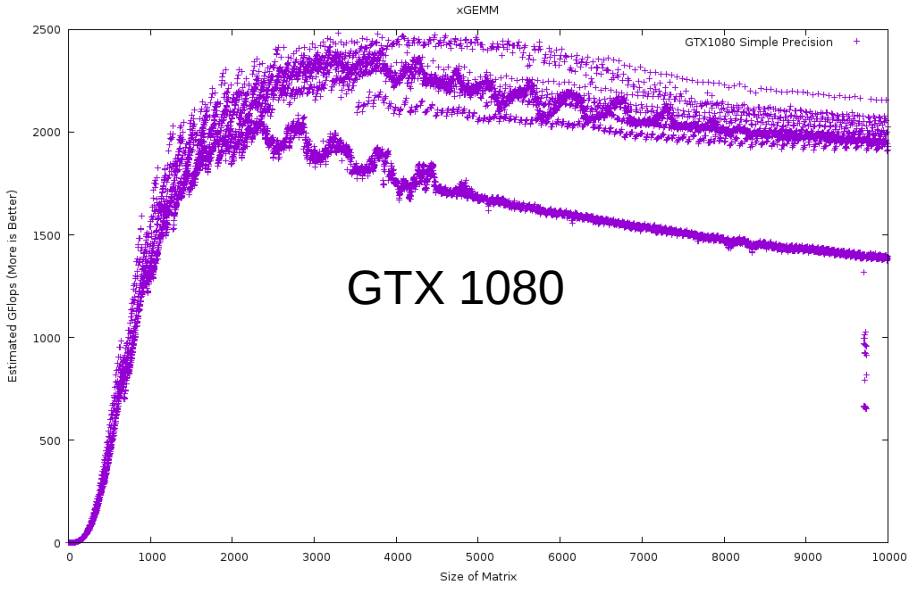
Déjà en 2010, étrange… Le comportement Tesla C1060...

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

Ce que Nvidia ne précise jamais…
xGEMM sur des tailles différentes...

Performance

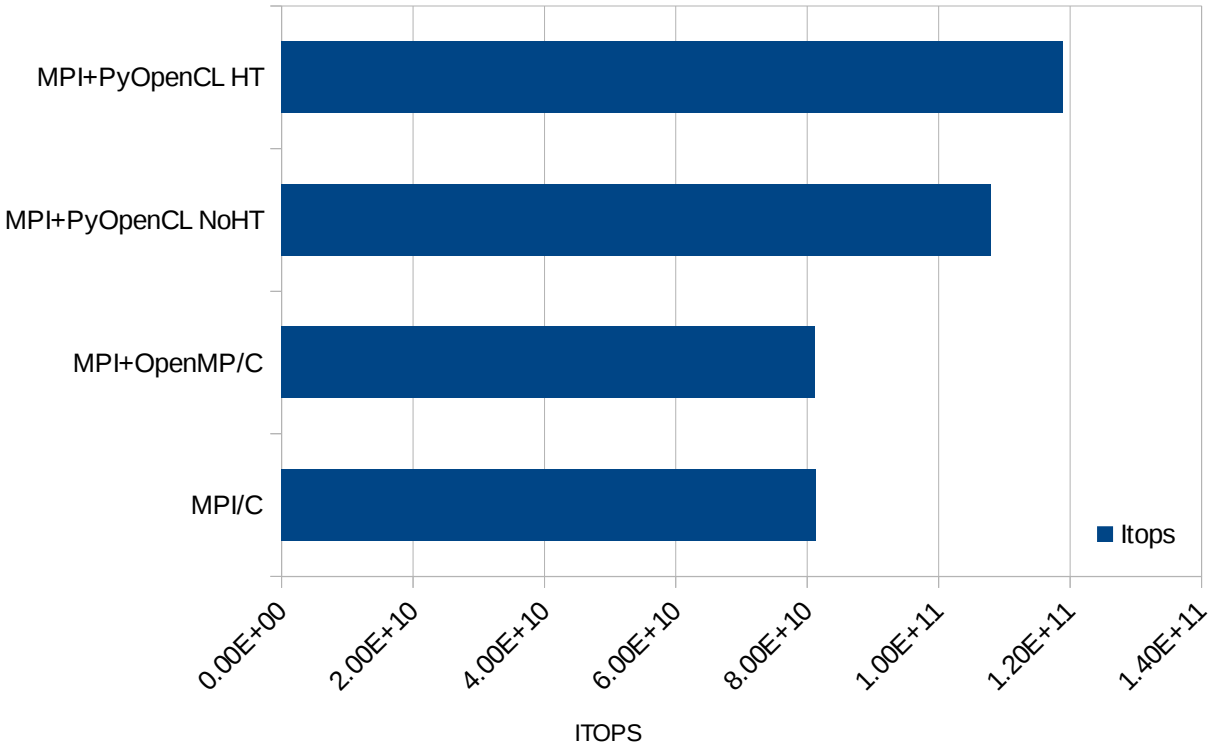




Taille

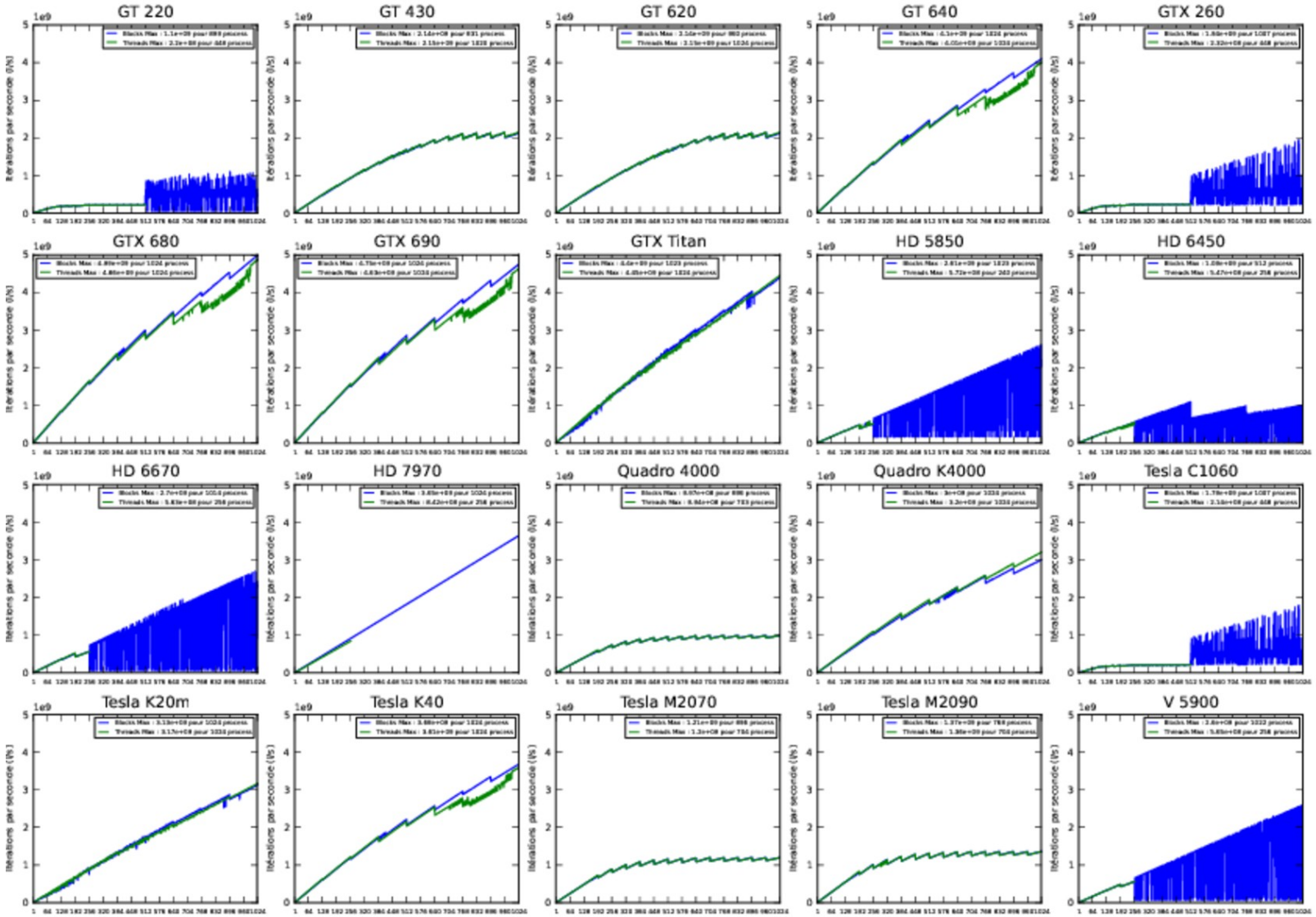
Oui, il y a la performance, mais dans certaines conditions...

Python est-il efficace ?
Une rapide comparaison avec GNU

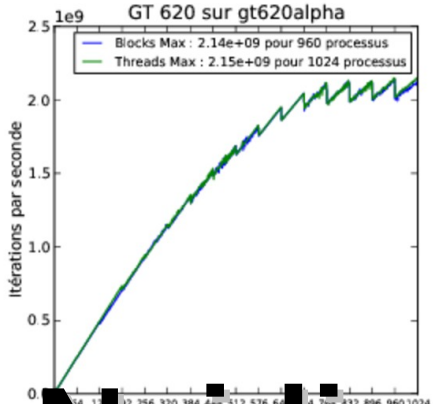


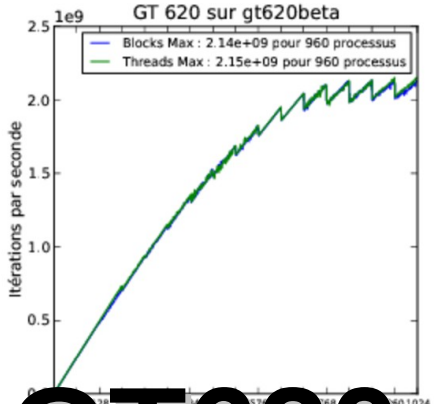
- 3 implémentations (2 hybrides)

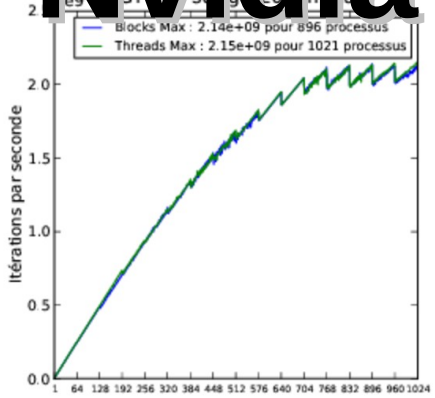
Comportement de 20 (GP)GPU vs PR

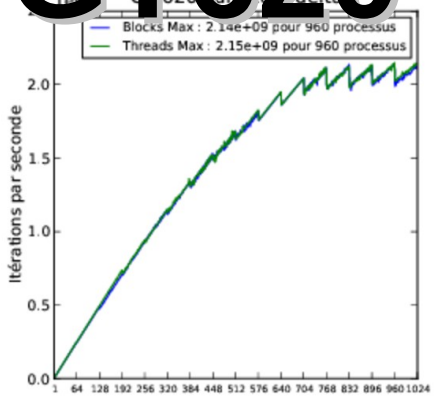


Enveloppes de parallélisme reproductibles ? Pour deux cartes spécifiques...

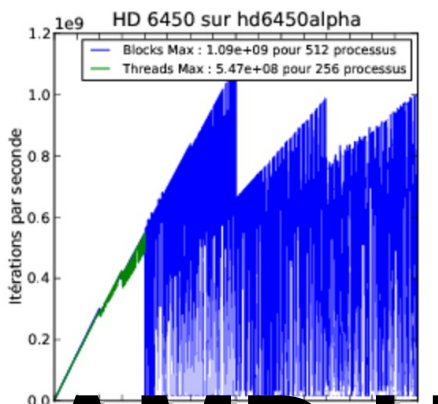


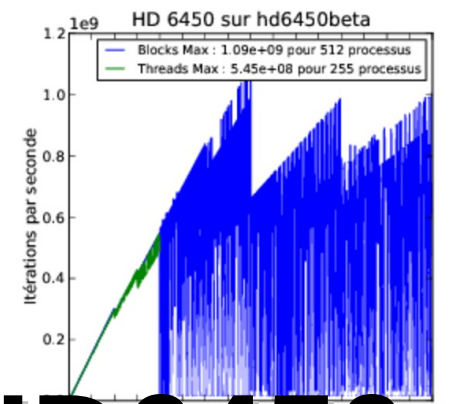


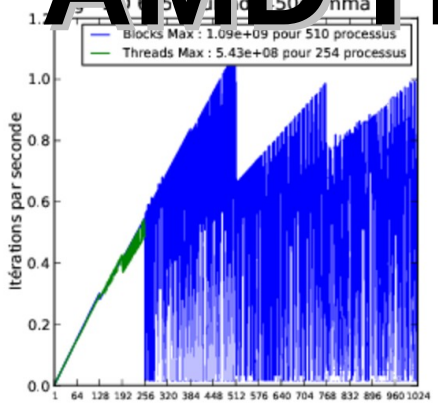


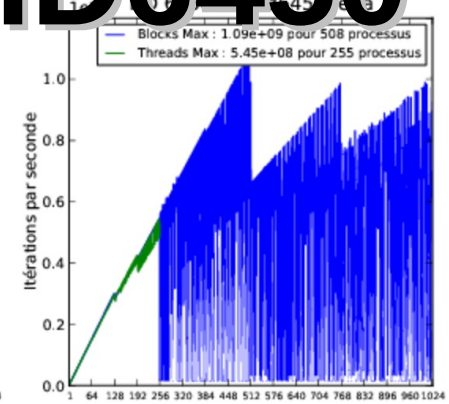


Nvidia GT620



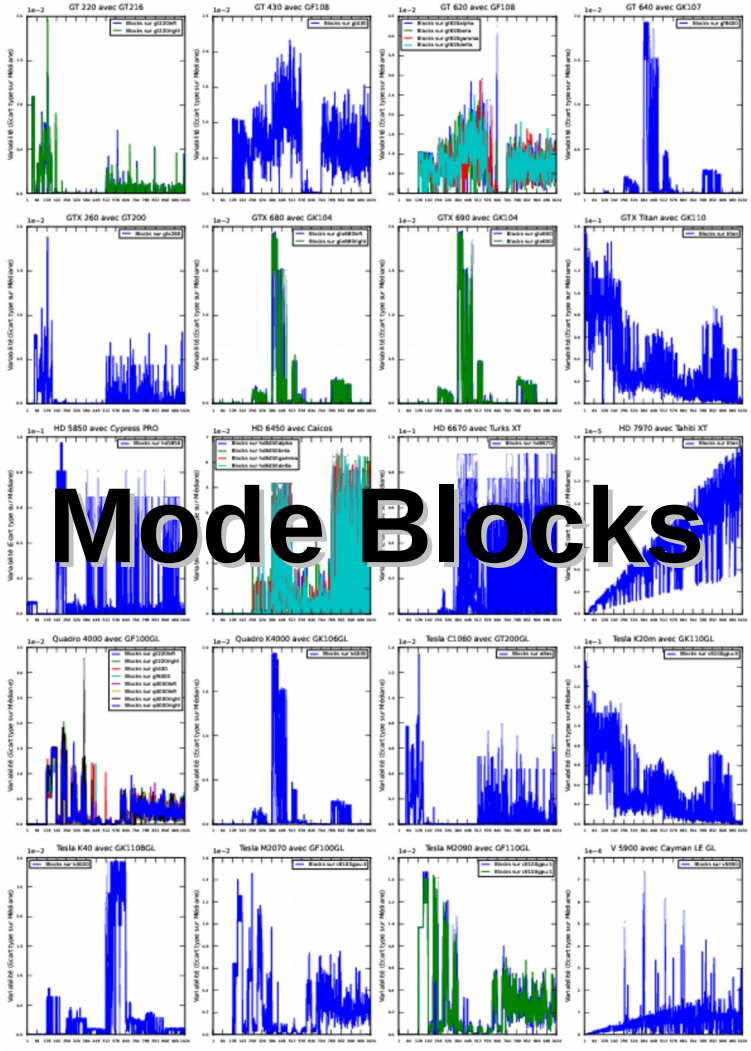




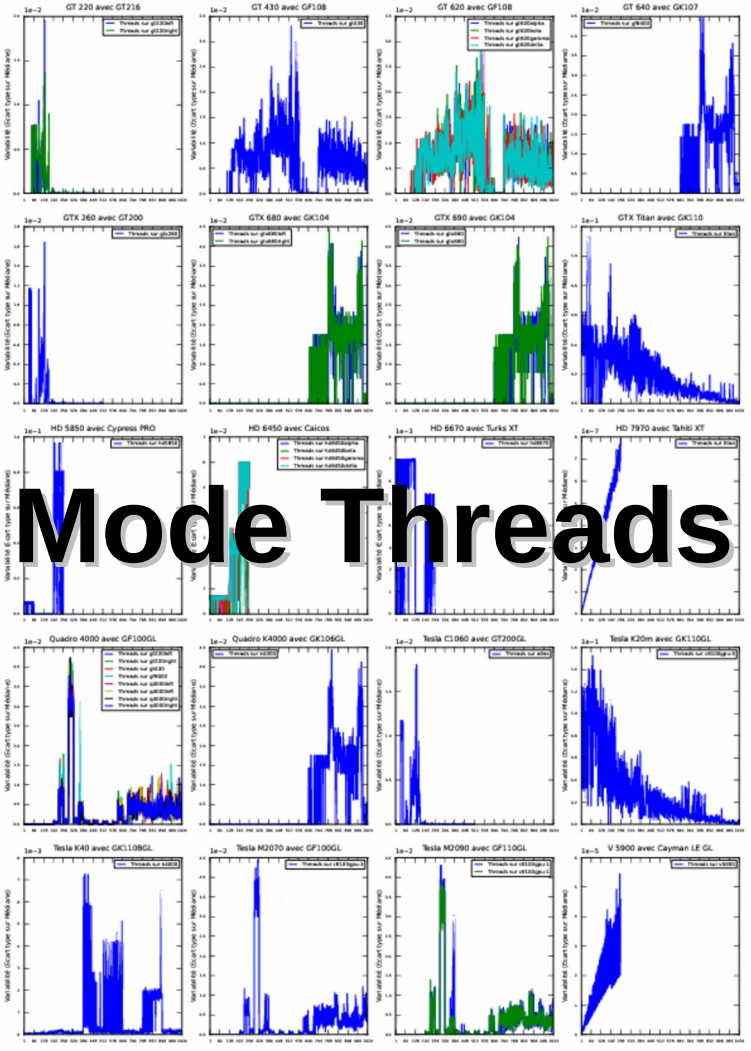


AMD HD6450

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



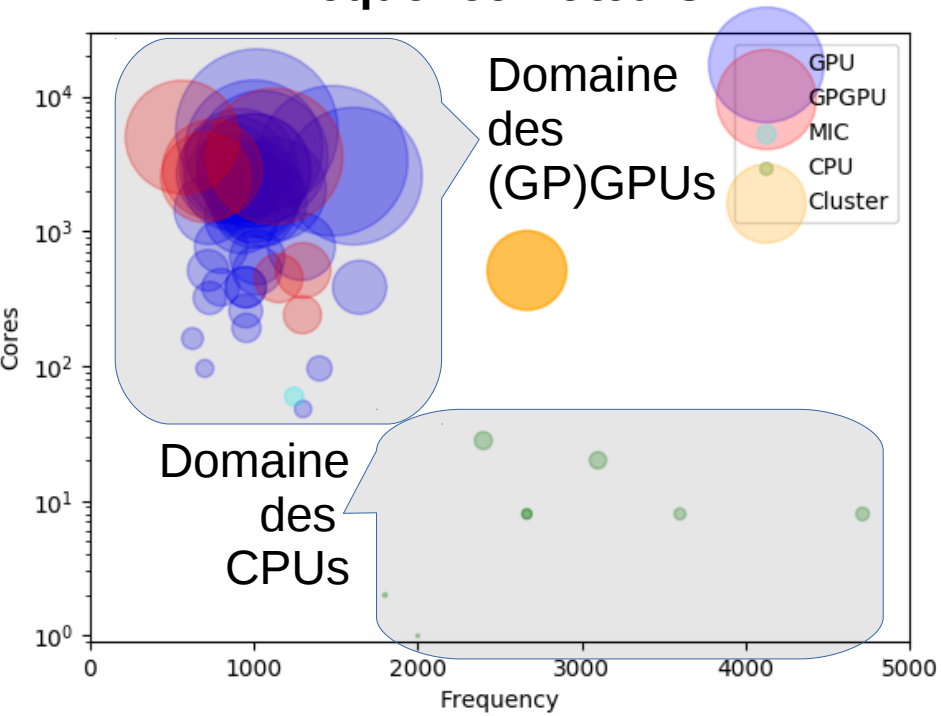
Mode Blocks



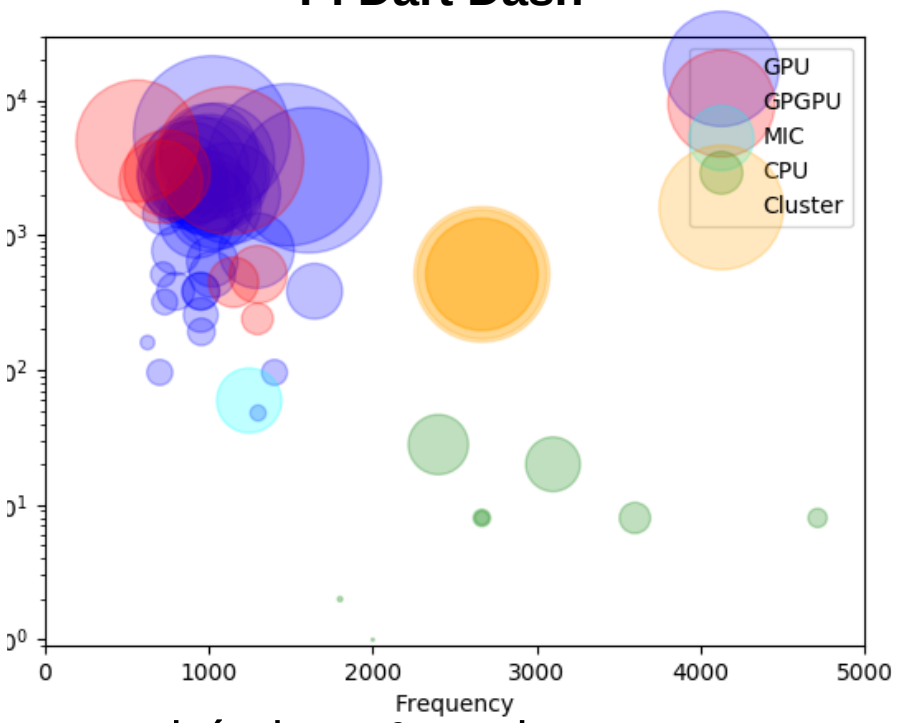
Mode Threads

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

Performance Théorique
Fréquence * cœurs



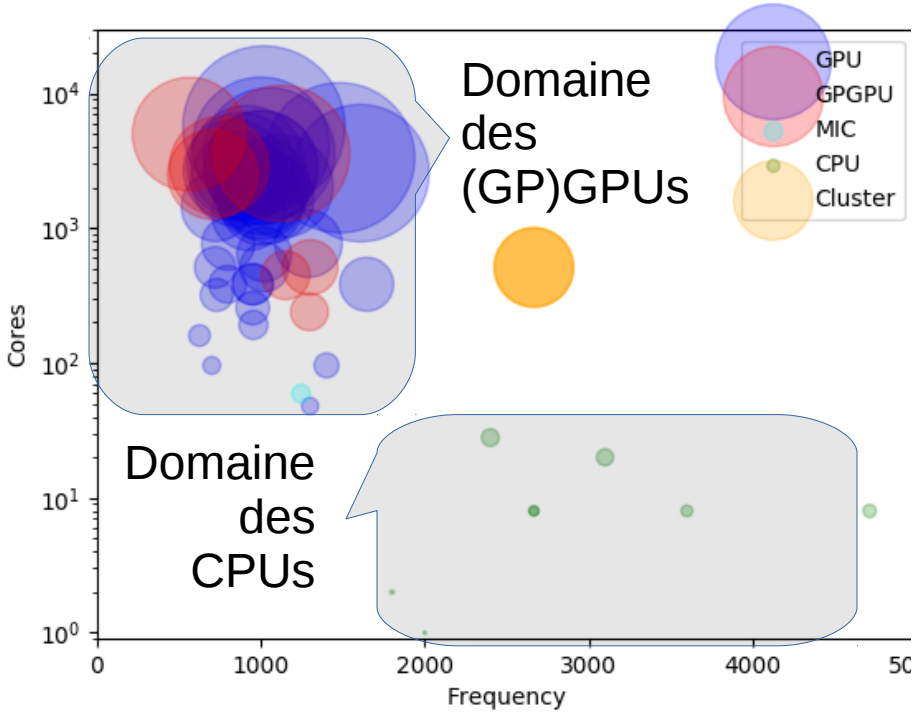
Performance en 32 bits
“Pi Dart Dash”

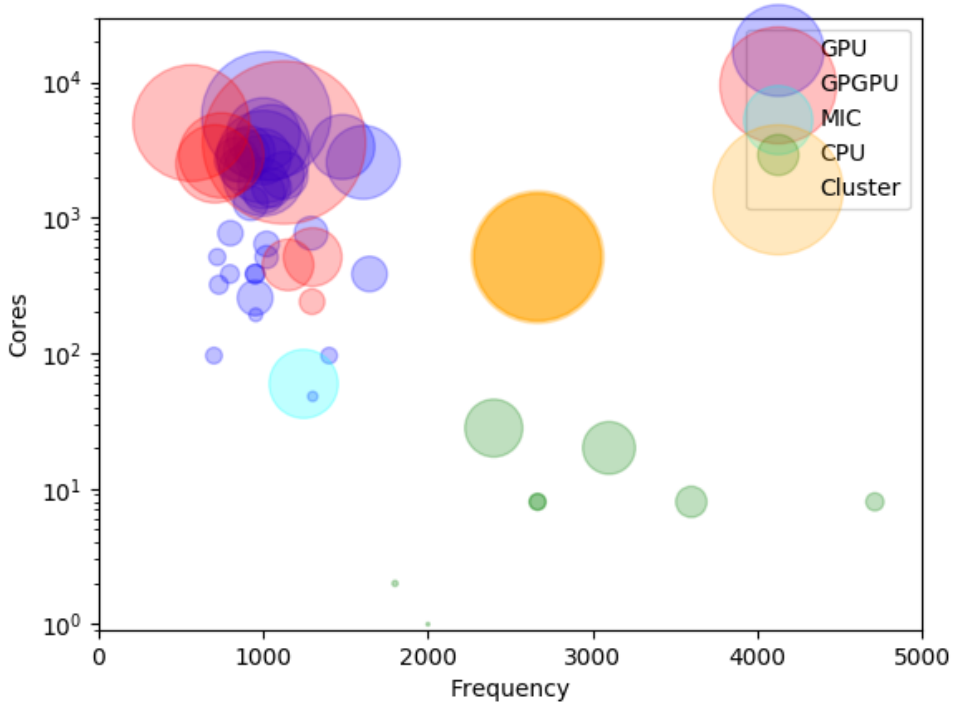


- Sur un GPU, cohérence entre performances théorique & pratique
- Sur un CPU, performance relative meilleure

La performance en informatique

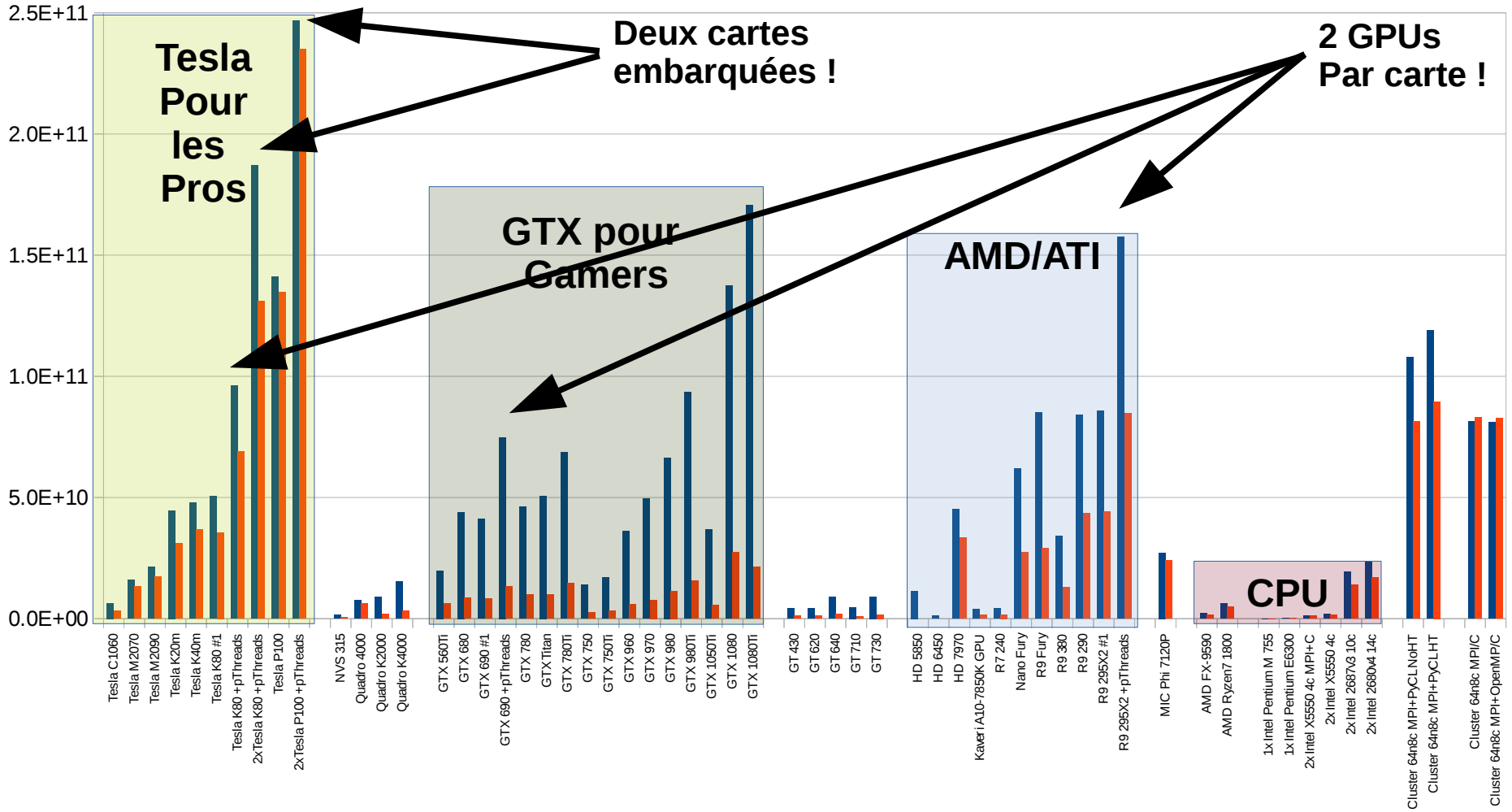
Question de battement, de cœurs, et de bits !



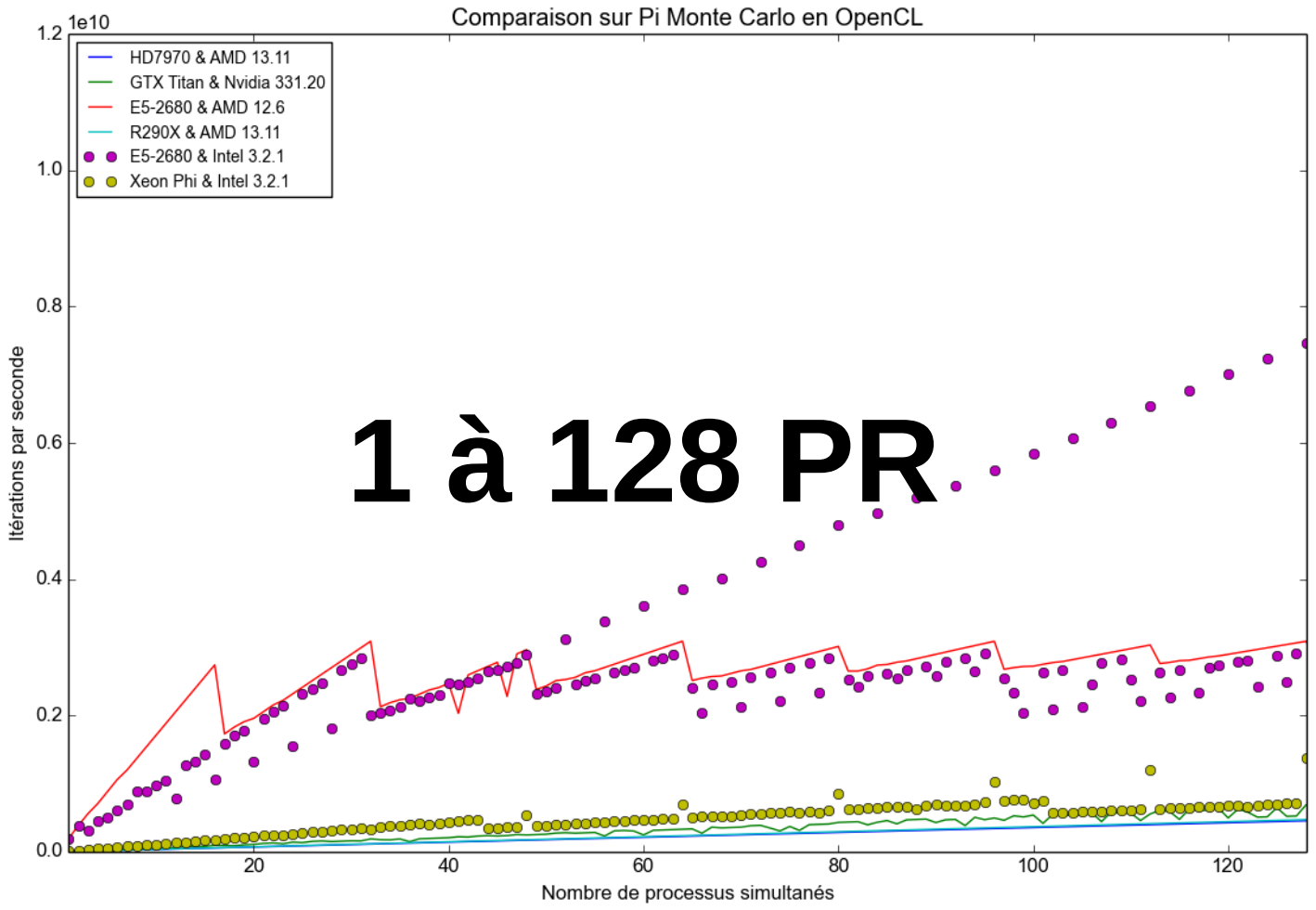


- Sur un GPU, baisse très sensible des performances pour les GPU
- Sur un CPU, performance relative encore meilleure

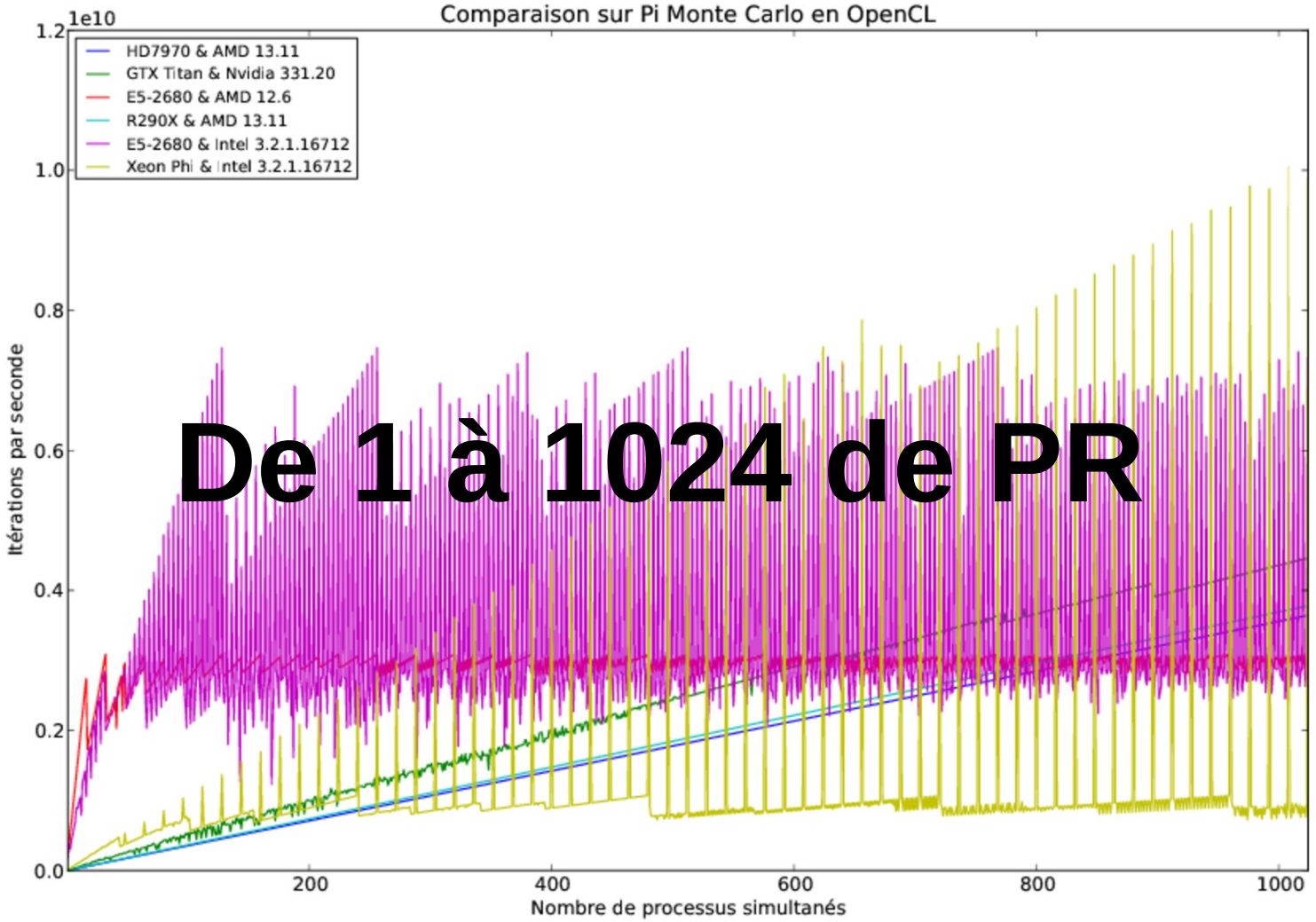
Le match Pi Monte Carlo Python vs C & CPU vs GPU vs Phi



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

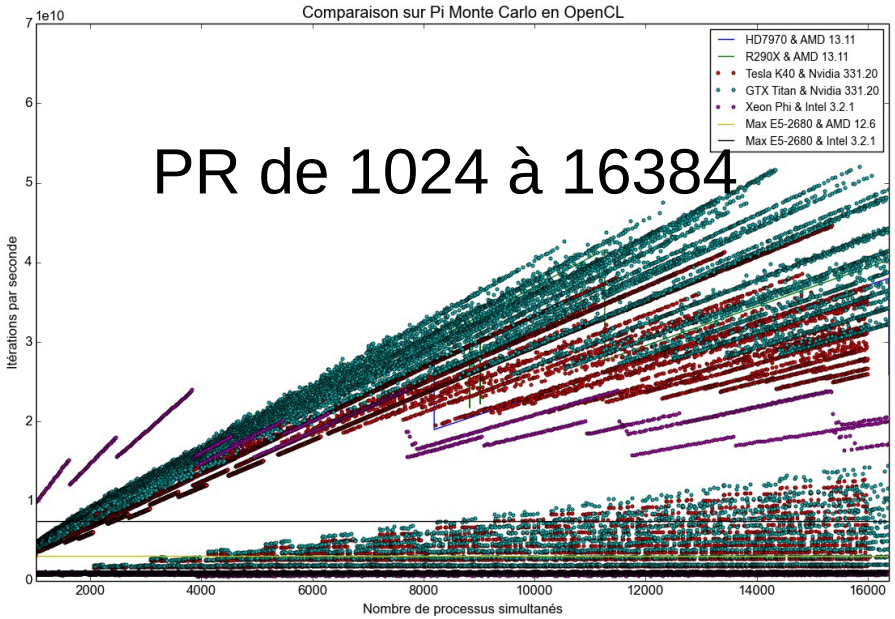


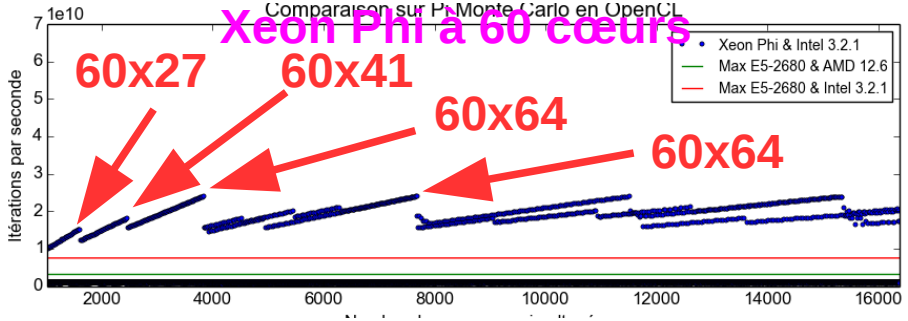
Les processeurs toujours efficaces
Le Xeon Phi « sort du bois »

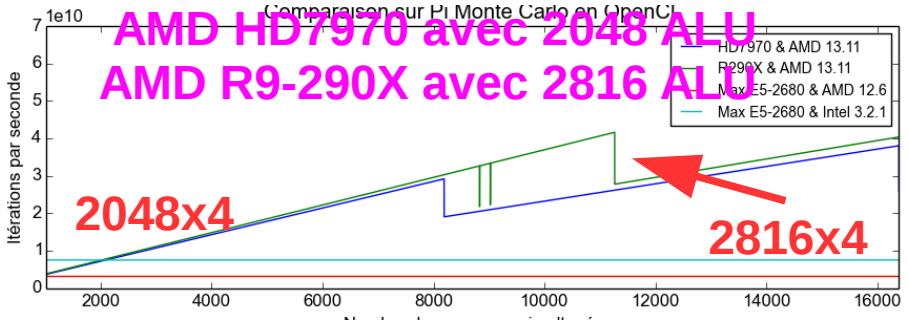


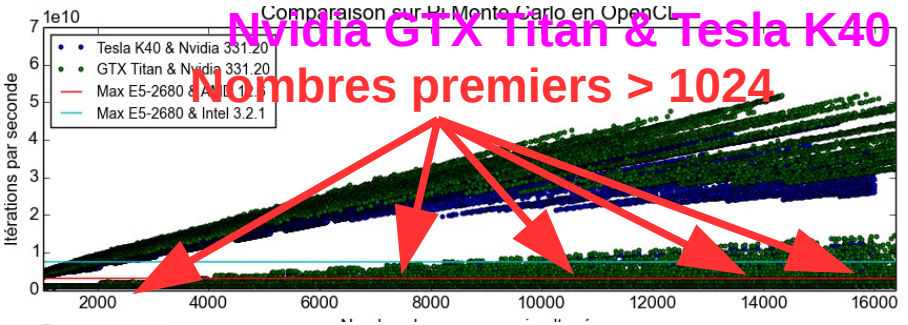
De 1 à 1024 de PR

Exploration profonde de PR ParaRégime de 1024 à 16384

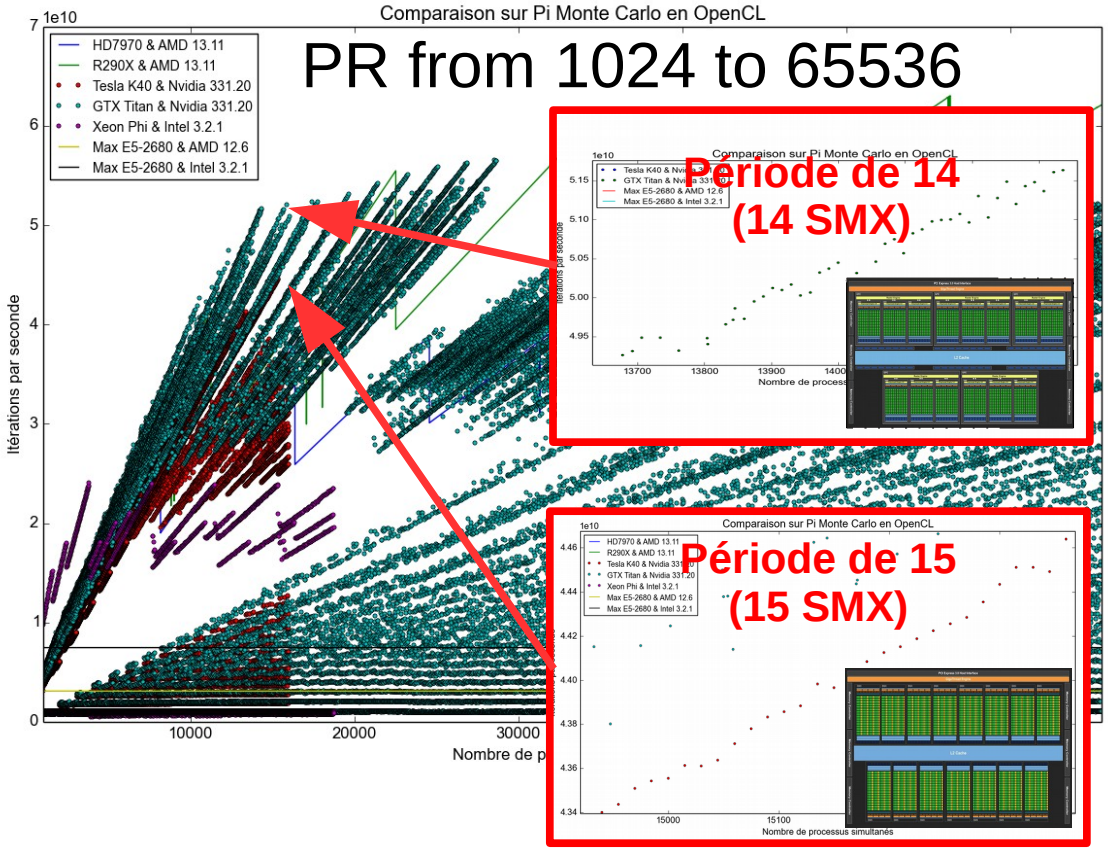




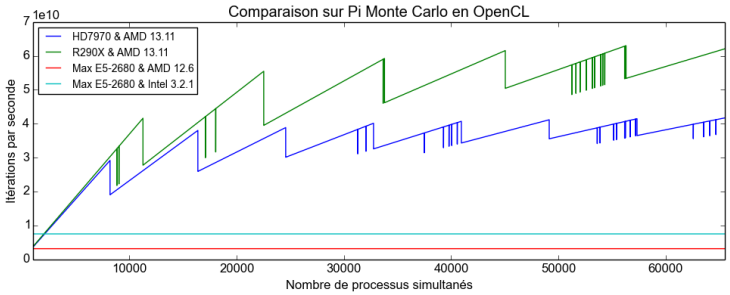




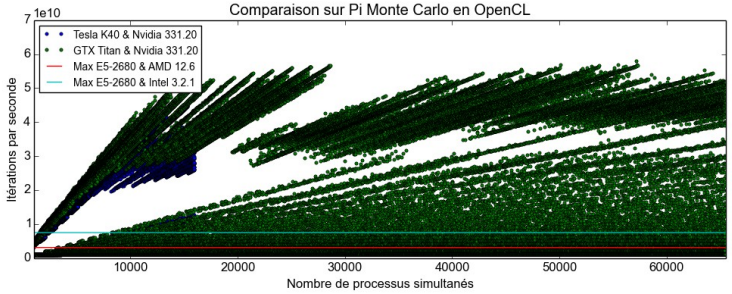
Investiguer la structure interne Par l'exécution de Pi Dart Dash



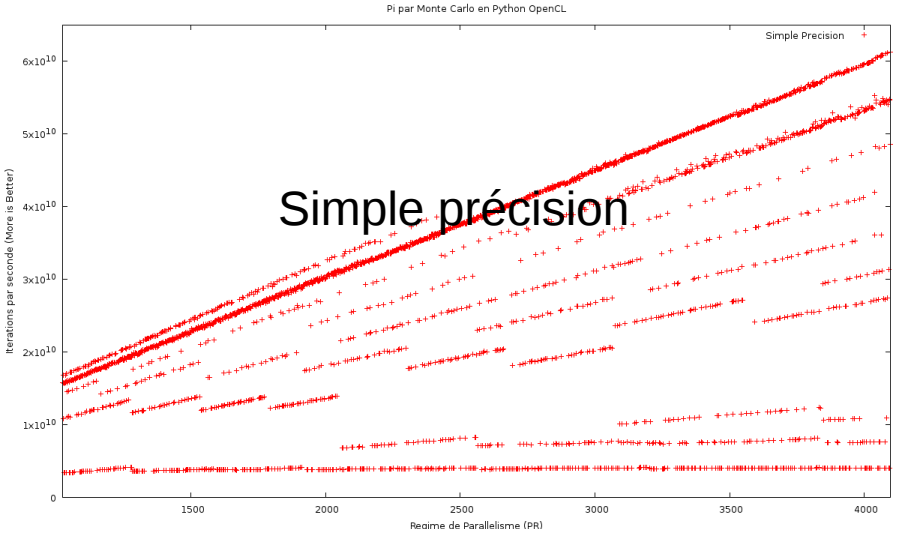
AMD HD7970 & R9-290
Longue Période : 4x nombre d'ALU

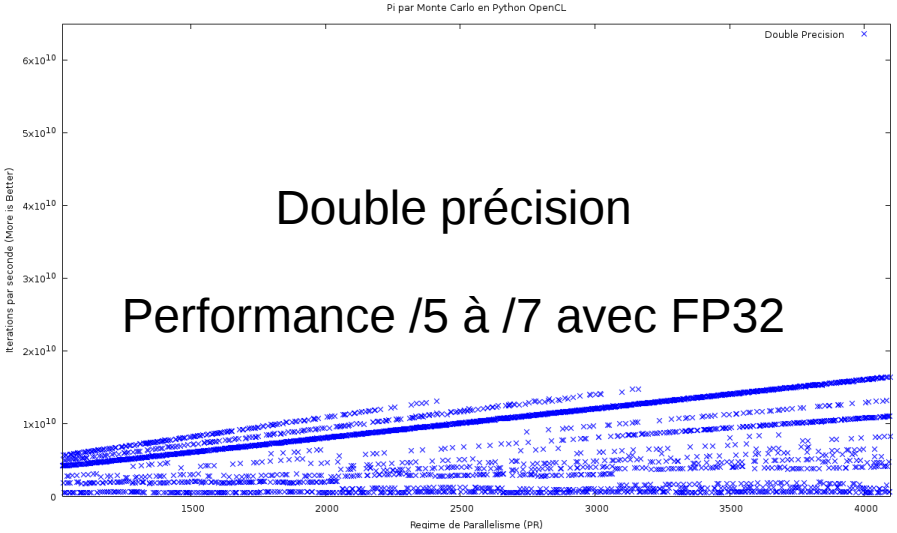


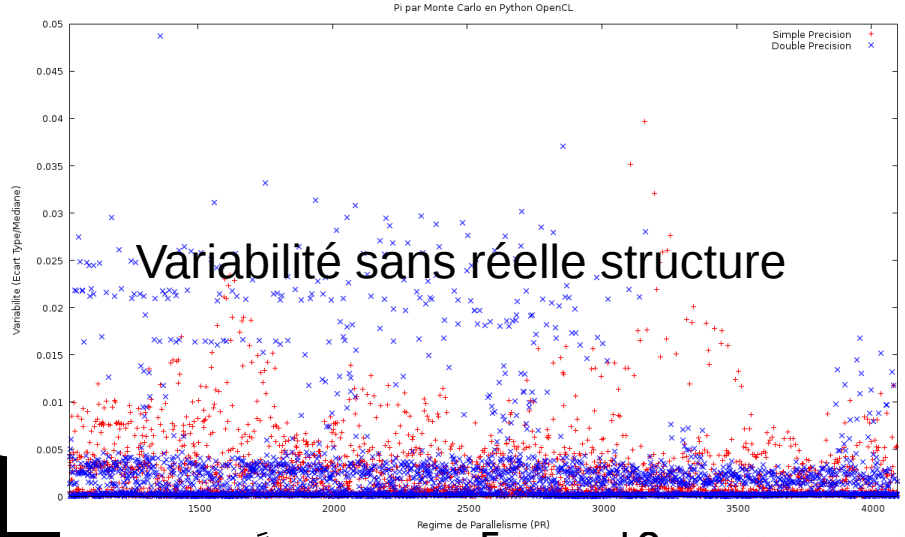
Nvidia GTX Titan & Tesla K40
Courte Période : nombre d'unités SMX

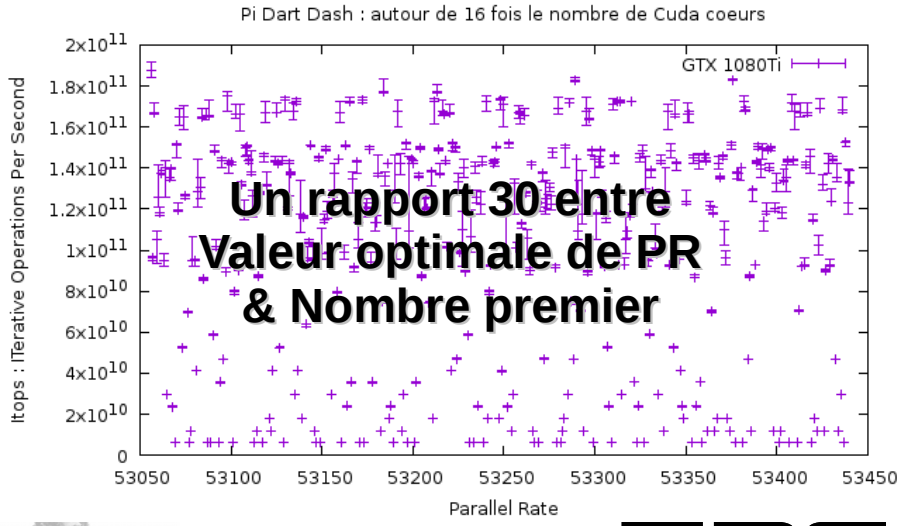


Et les dernières Nvidia GTX1080(Ti) Toujours affectées de bizarreries ?

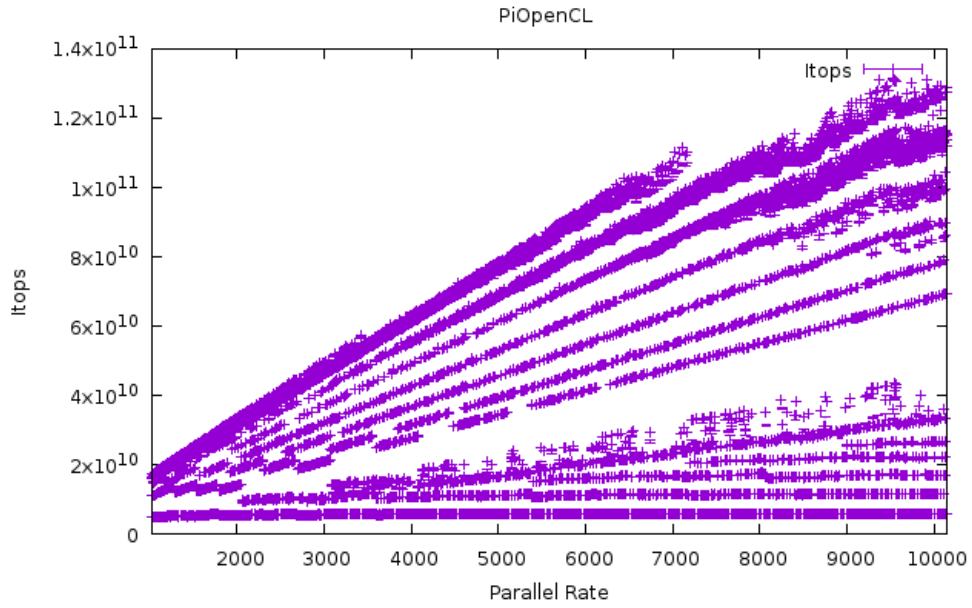


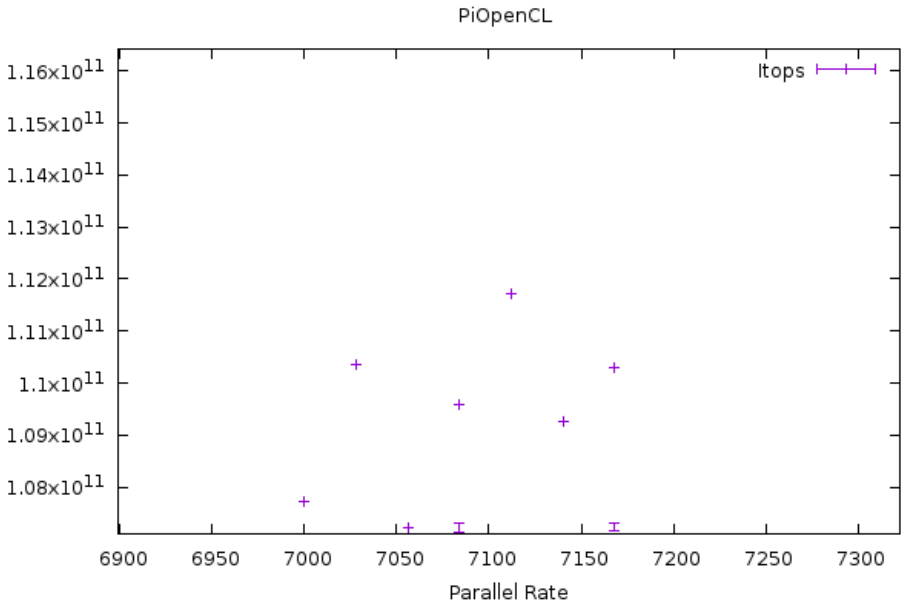






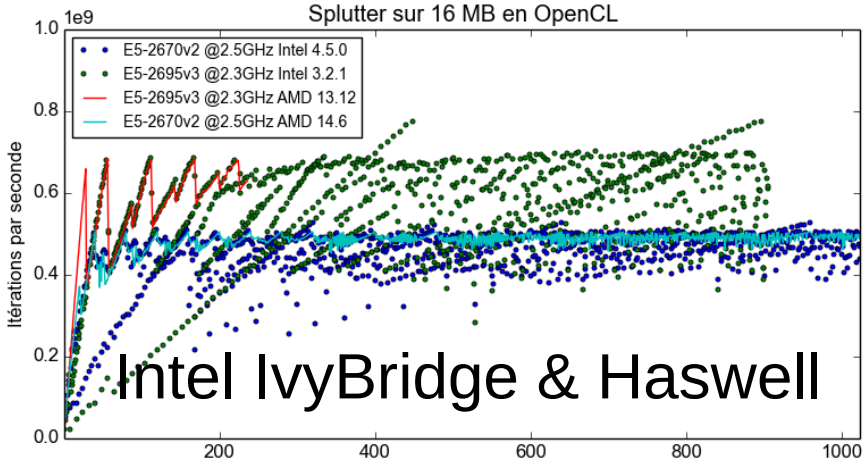
Et en passant sur un PiOpenCL en C pur ?

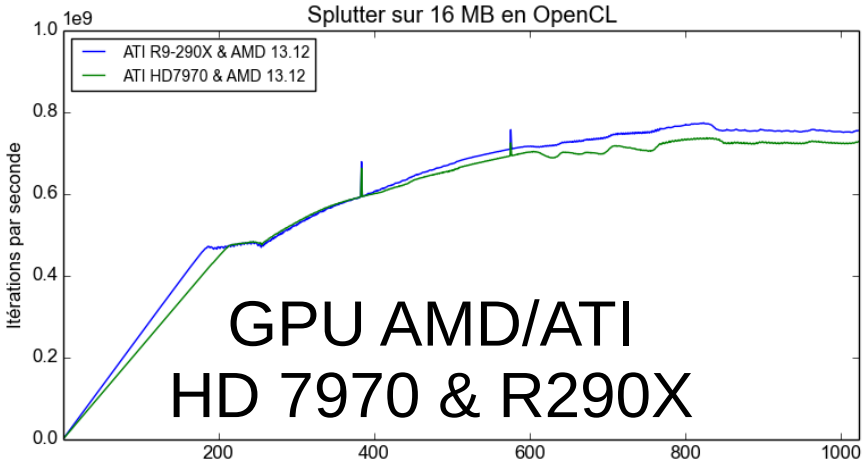


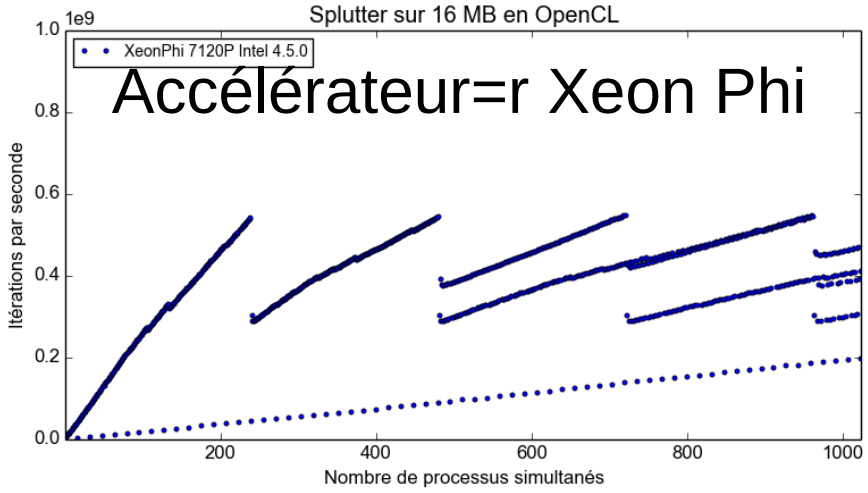


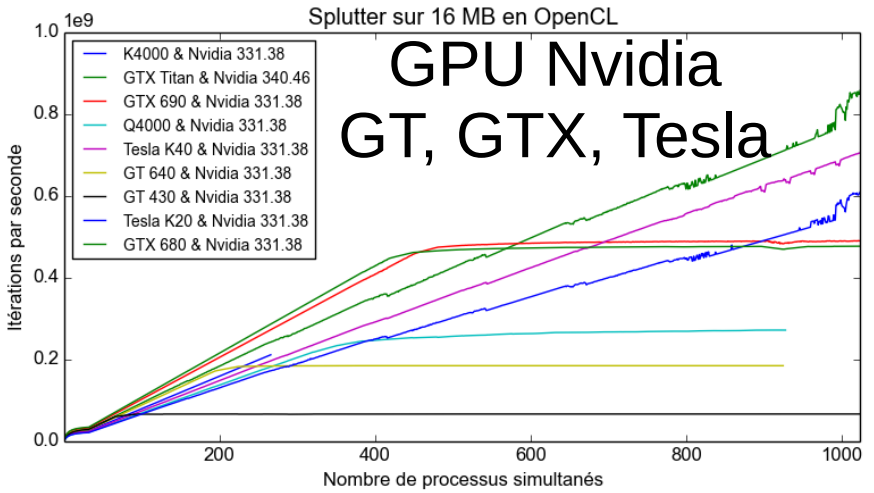
- Les mêmes détections de nombres premiers
- Les mêmes max (x2 le nombre de shaders)

Un test « memory bound »
Le « postillonneur » en OpenCL





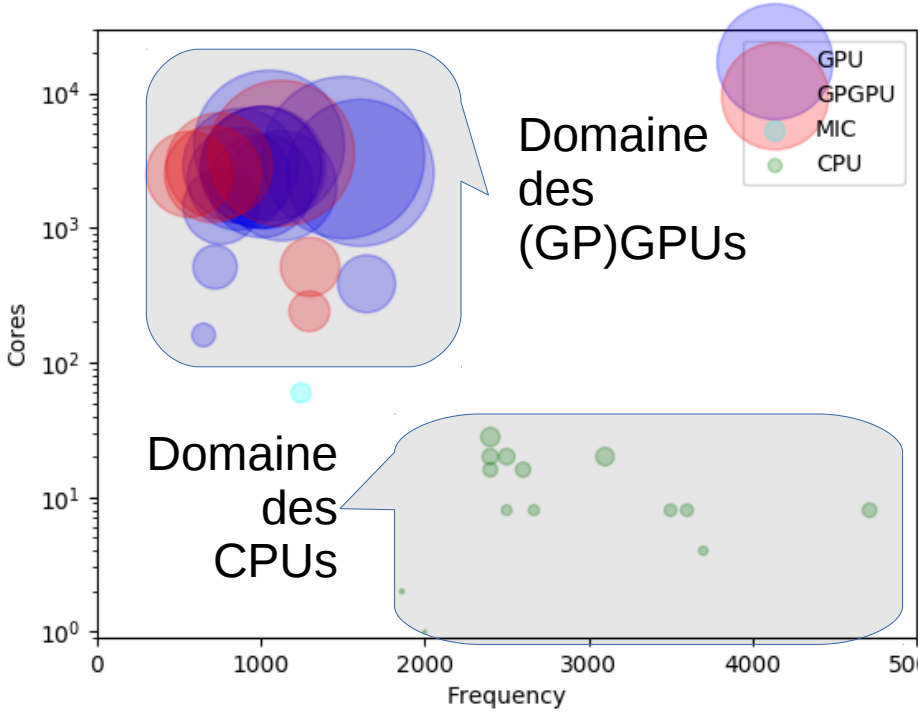


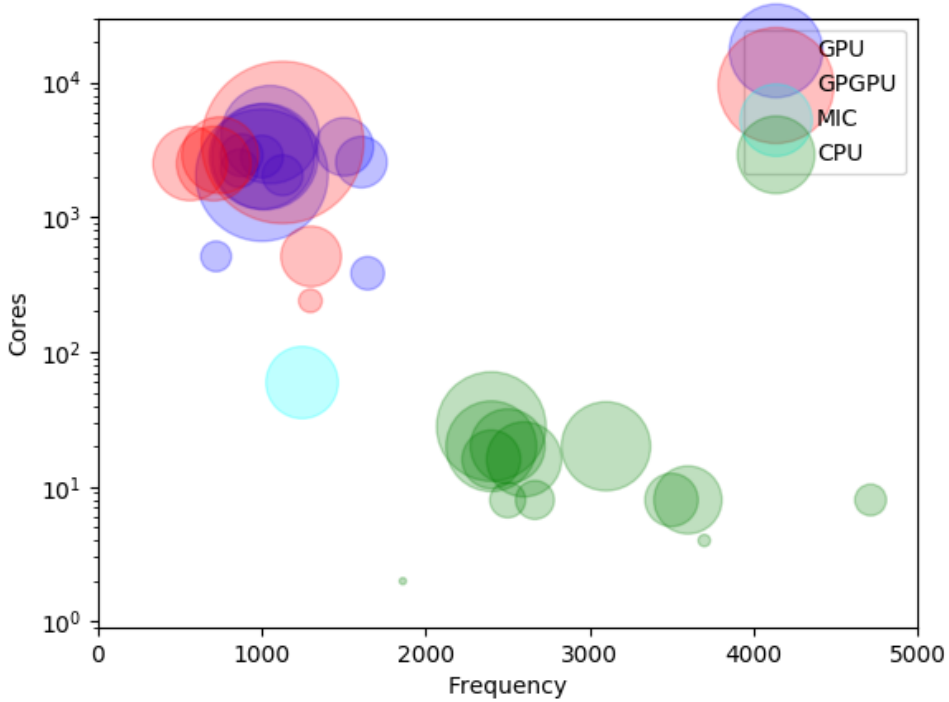


Retour à la physique
Un code newtonien N-corps...

- 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 Explicit, 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

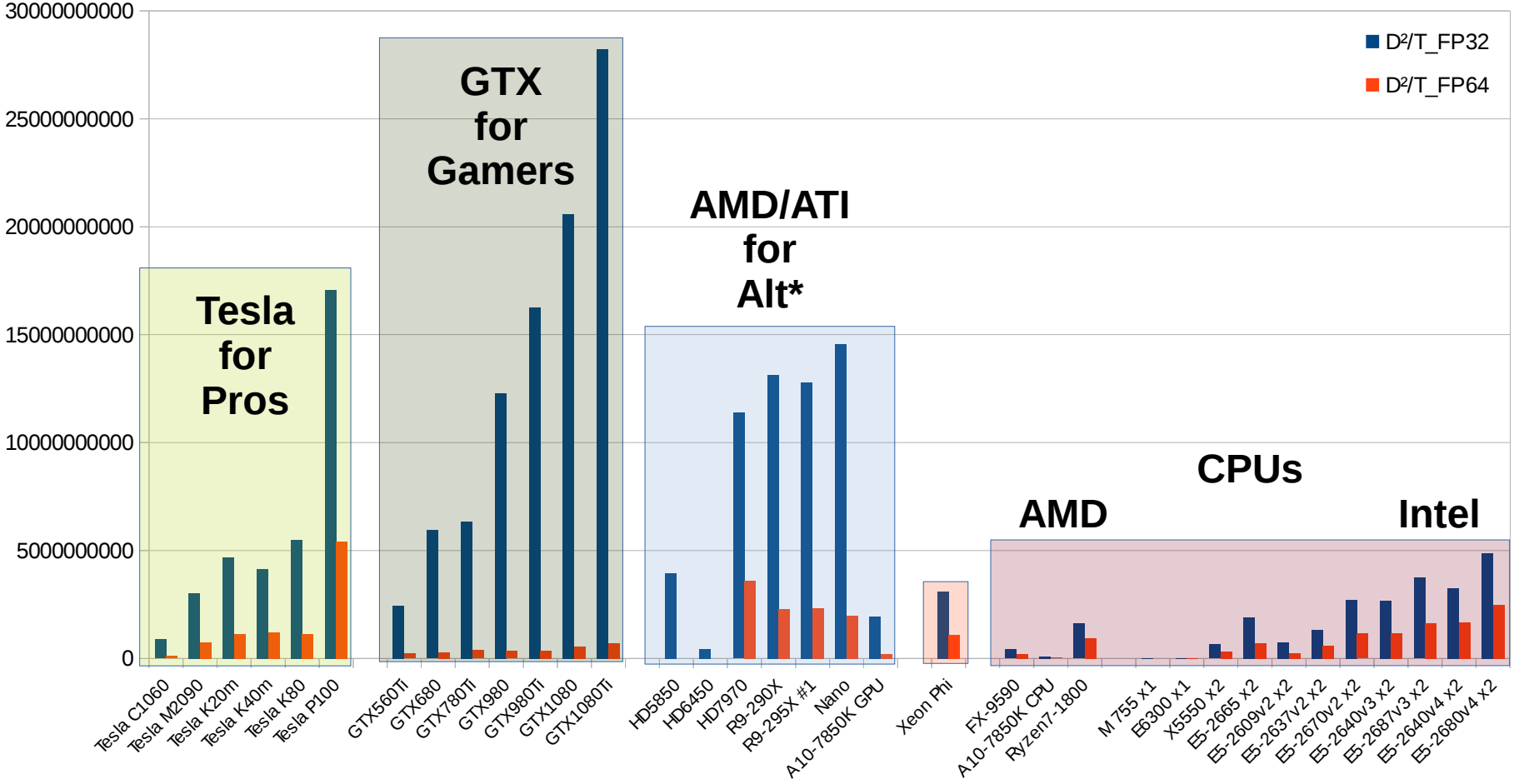
La performance en informatique
Pour un code plus physique...



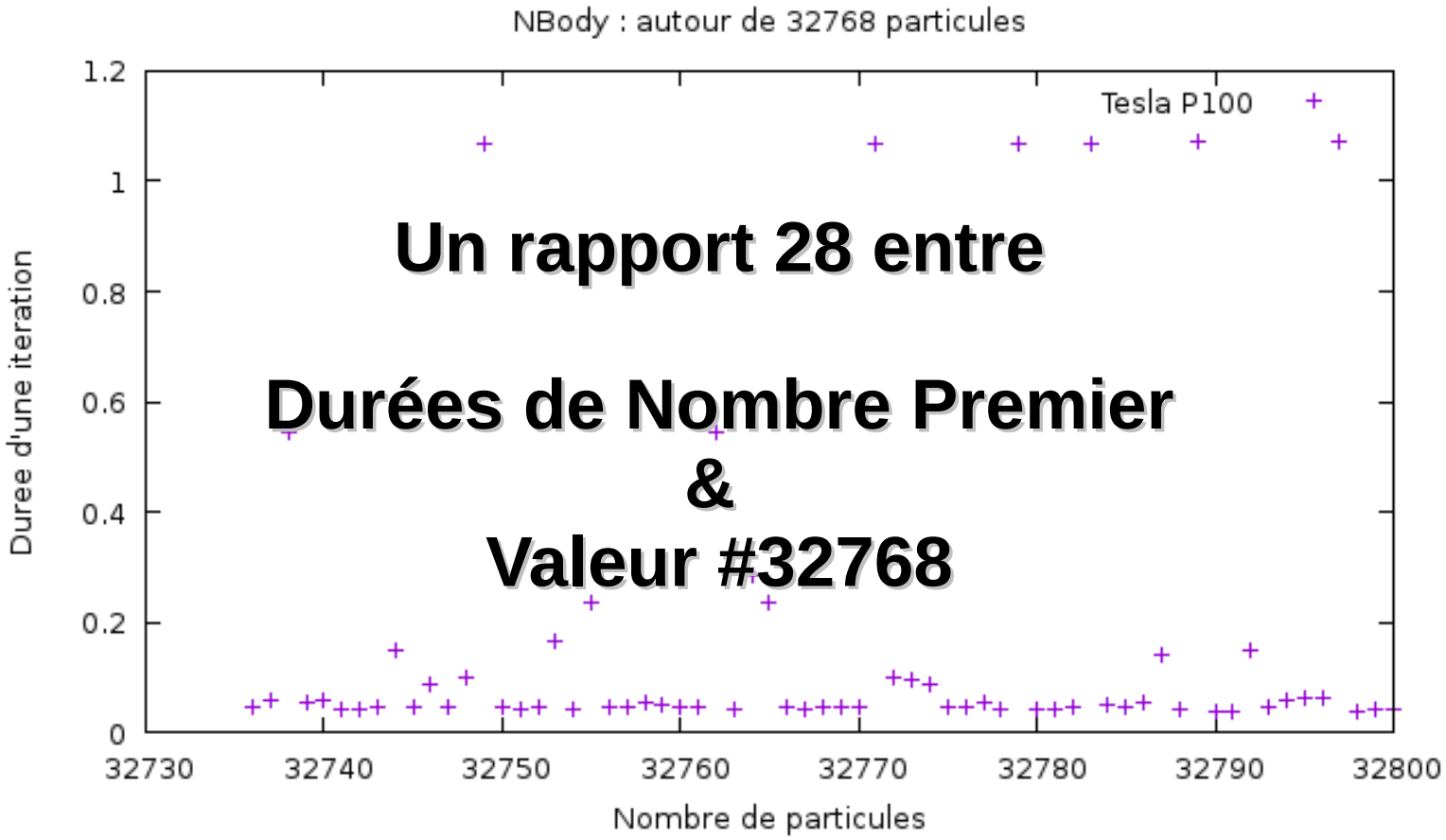


- Sur un GPGPU, performance stable
- Sur un GPU, baisse drastique de la performance (division par 20)
- Sur un CPU, performance relative meilleure

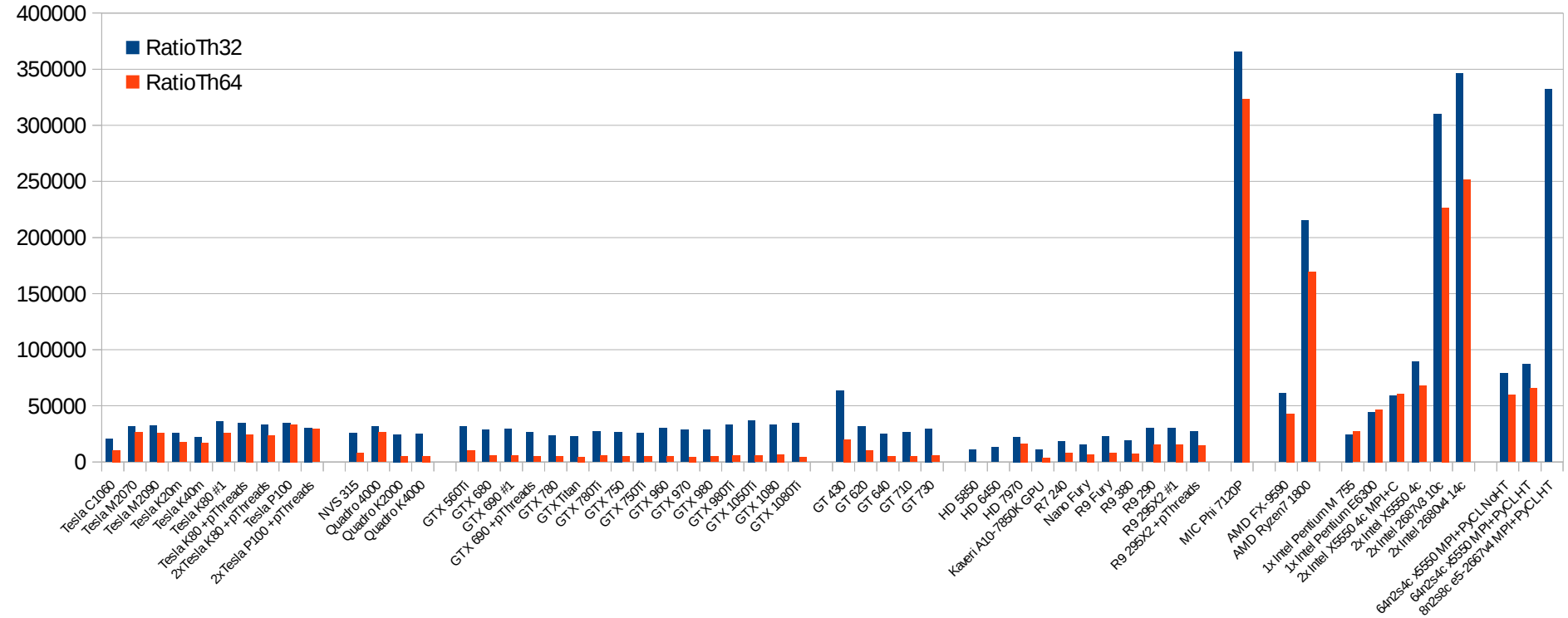
Modèle N-corps « naïf »
Une comparaison classique...



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

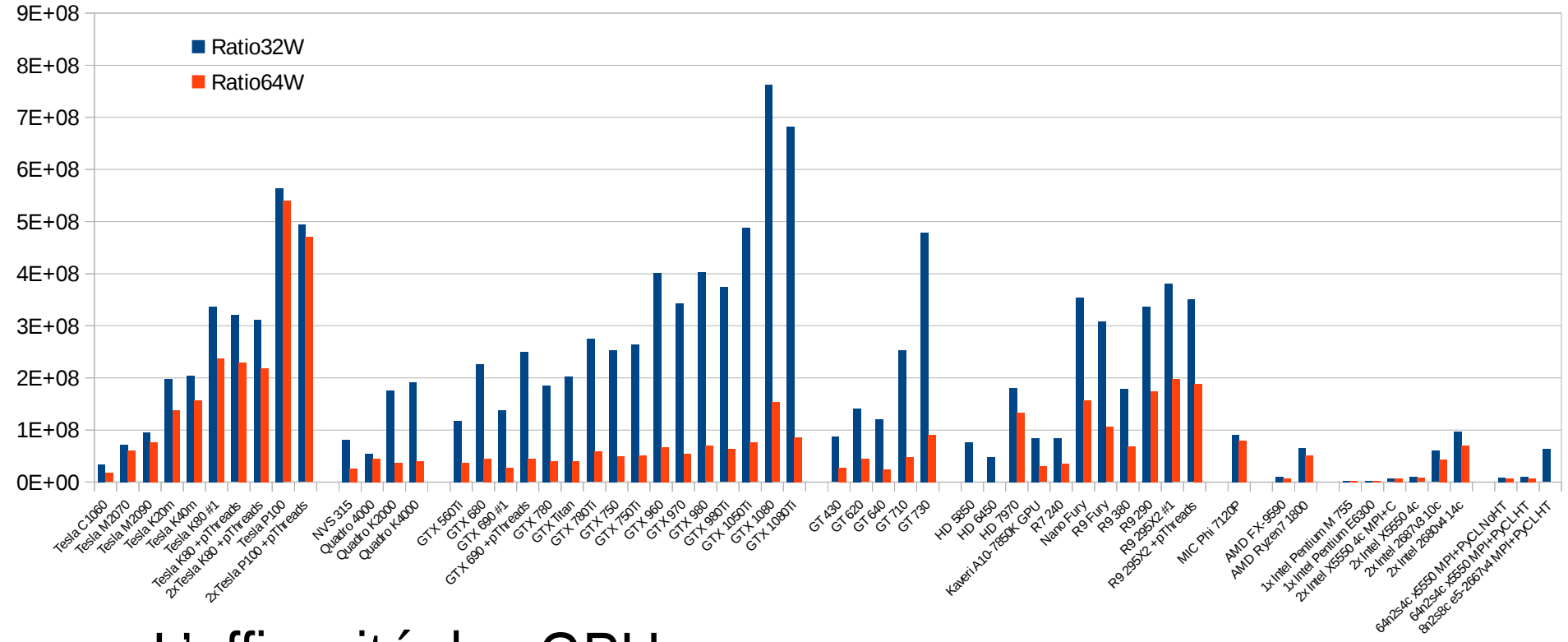


Performances normalisées : quel ratio ?
Pi Dart Dash / Fréquence*Nb cœurs



- Pas d'évolution significative pour les (GP)GPUs
- Ratio largement croissant pour les CPU : Intel & AMD

Performances par Watt
Pi Dart Dash / TDP



- L'efficacité des GPU
- La progression des CPU

Introduction à la conclusion : IT : le nouveau monde de la complexité

- Compliqué : de « cum plicare », « plier ensemble »
  - Descartes : « Le tout est la somme des parties. »
- Complexe : de « cum plexus », « tisser ensemble »
  - Immense nombre d'interaction, non linéarité, émergence, ...
- Les ressources de calcul sont des systèmes
  - Un Operating System porte au moins 200 processus en tâche de fond
  - Les cœurs CPU changent leur fréquence & voltage tout le temps, start/stop, …
  - La DRAM change sa fréquence tout le temps
  - Les éléments réseau exploitent tous des mécanismes avec accès aléatoire

Modèle OSI & Loi d'Amdahl Évolutions & perspectives

- Modèle OSI : La couche du dessous comme un service
  - ignorance de l'infrastructure socle : suicide l'étude de la scalabilité...
- Loi d'Amdahl : Ne dépend que de T_1 et p
  - Ce n'est jamais représentatif de ce qui s'exécute...
- Loi de Mylq : ajout d'un terme du premier order (et second)
  - Finalement assez représentatif des comportements observés, avec vigilance
  - Utilisable pour évaluer la scalabilité et prédire une performance
- Dans un nœud, aucune loi ne fonctionne...
  - Et c'est bien pire pour les GPU et les accélérateurs !

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

- Ce qu'il faut retenir :
  - Dans une machine, en 2017, 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
  - 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 !

Ateliers 3IP
Donnez vos vieilles machines !

- 3IP : Introduction Inductive à L'informatique et au parallélisme
- Constat :
  - Rares sont les formations qui partent du matériel
- Objectif :
  - Mieux appréhender les usages par une connaissance du matériel
- Méthode :
  - Manipulation de composants informatiques
- Contact sur : emmanuel.quemener@ens-lyon.fr