Graphical Processing Units : Can we really entrust the compiler to reach the raw performance ?

Emmanuel Quemener
IR CBP

Starting with a tiny joke !

- How do you call people speak 3 languages ?
  - Trilingual people !
- How do you call people speak 2 languages ?
  - Bilingual people !
- How do you call people speak 1 language ?
  - French people !

I'm french :

if I twist your eardrums, I apologize...

Plan of talk...

- What's Blaise Pascal Center ?
  - What are the facilities provided for developing, learning, exploring in IT ?
- Where are we (now) ?
  - Raw processing power is NOT longer inside the CPU
- Where do we want to go (tomorrow) ?
  - What can we expect at most to reduce the Elapsed time...
- How can we go ?
  - Compiler ? High level librairies ? Hybrid approach ?

What's Blaise Pascal Center ?

Director : Pr Ralf Everaers

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

Multinodes TechBenchs : 9 clusters 116 nodes, 4 IB speeds

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

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

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

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

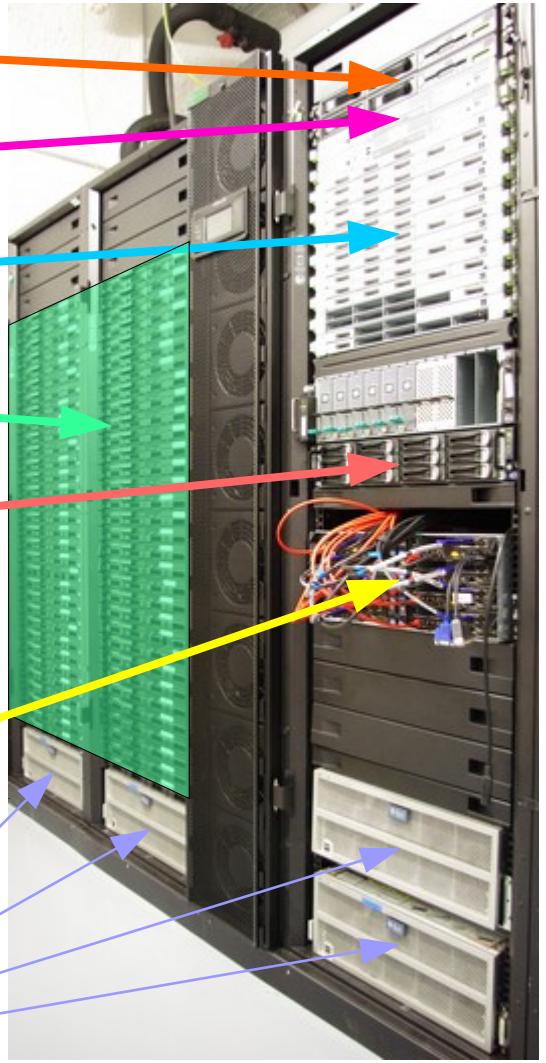
4 nodes Dell C6100 with Xeon X5650
48 physical cores HT @2666MHz
Interconnection Infiniband QDR 40 Gb/s
+ C410X with 4 GPGPU

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

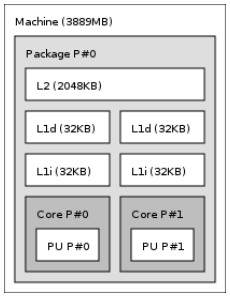
8 nodes HP SL230 with Xeon E5-2667
64 physical cores HT @2666MHz
Interconnection Infiniband FDR 56 Gb/s

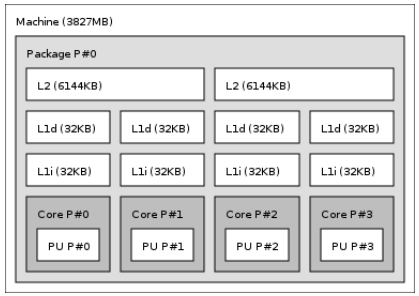
8 nodes Dell R410 with Xeon X5550
64 physical cores HT @2666MHz
Interconnection Infiniband DDR 20 Gb/s

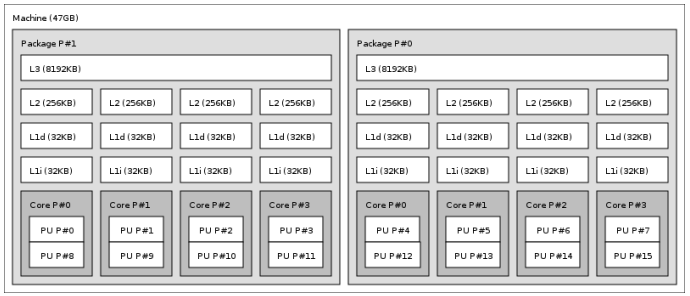
4 nodes Sun X4500 with AMD 285
16 physical cores @2400MHz
Interconnection Infiniband SDR 10 Gb/s

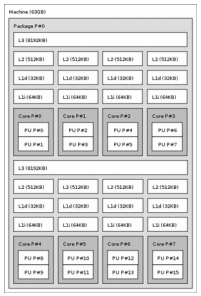


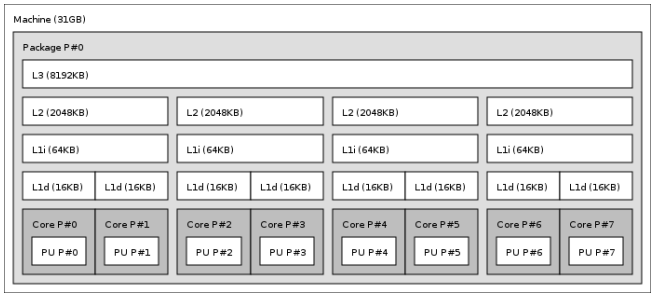
Multicores TechBenchs
From 2 to 28 cores: examples...

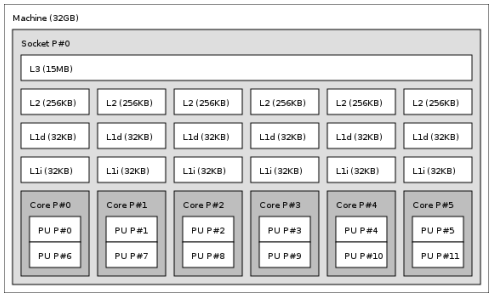


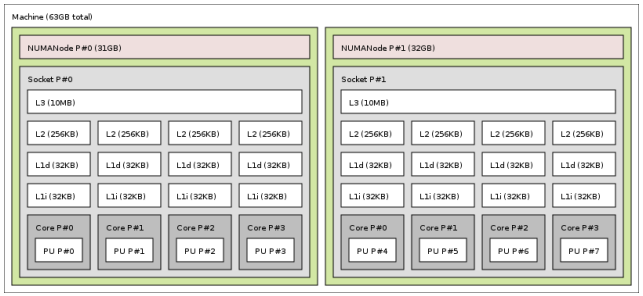


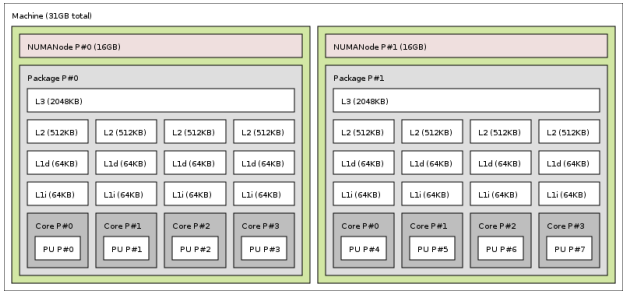


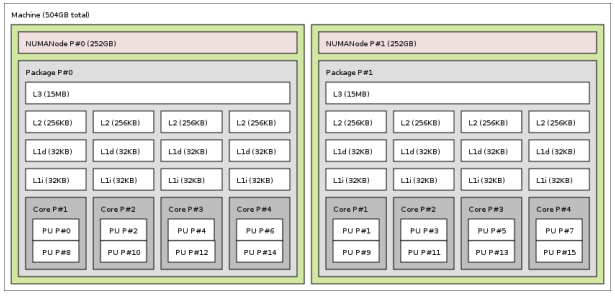












Multishaders TechBenchs (GP)GPU

72 different models...

GPU Gamer : 18

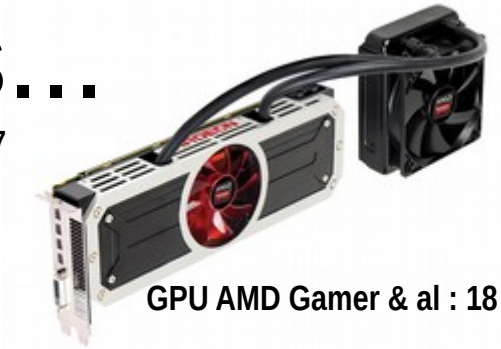
- Nvidia GTX 560 Ti
- Nvidia GTX 680
- Nvidia GTX 690
- Nvidia GTX Titan
- Nvidia GTX 780
- Nvidia GTX 780 Ti
- Nvidia GTX 750
- Nvidia GTX 750 Ti
- Nvidia GTX 960
- Nvidia GTX 970
- Nvidia GTX 980
- Nvidia GTX 980 Ti
- Nvidia GTX 1050 Ti
- Nvidia GTX 1060
- Nvidia GTX 1070
- Nvidia GTX 1080
- Nvidia GTX 1080 Ti
- Nvidia RTX 2080 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 : 27

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



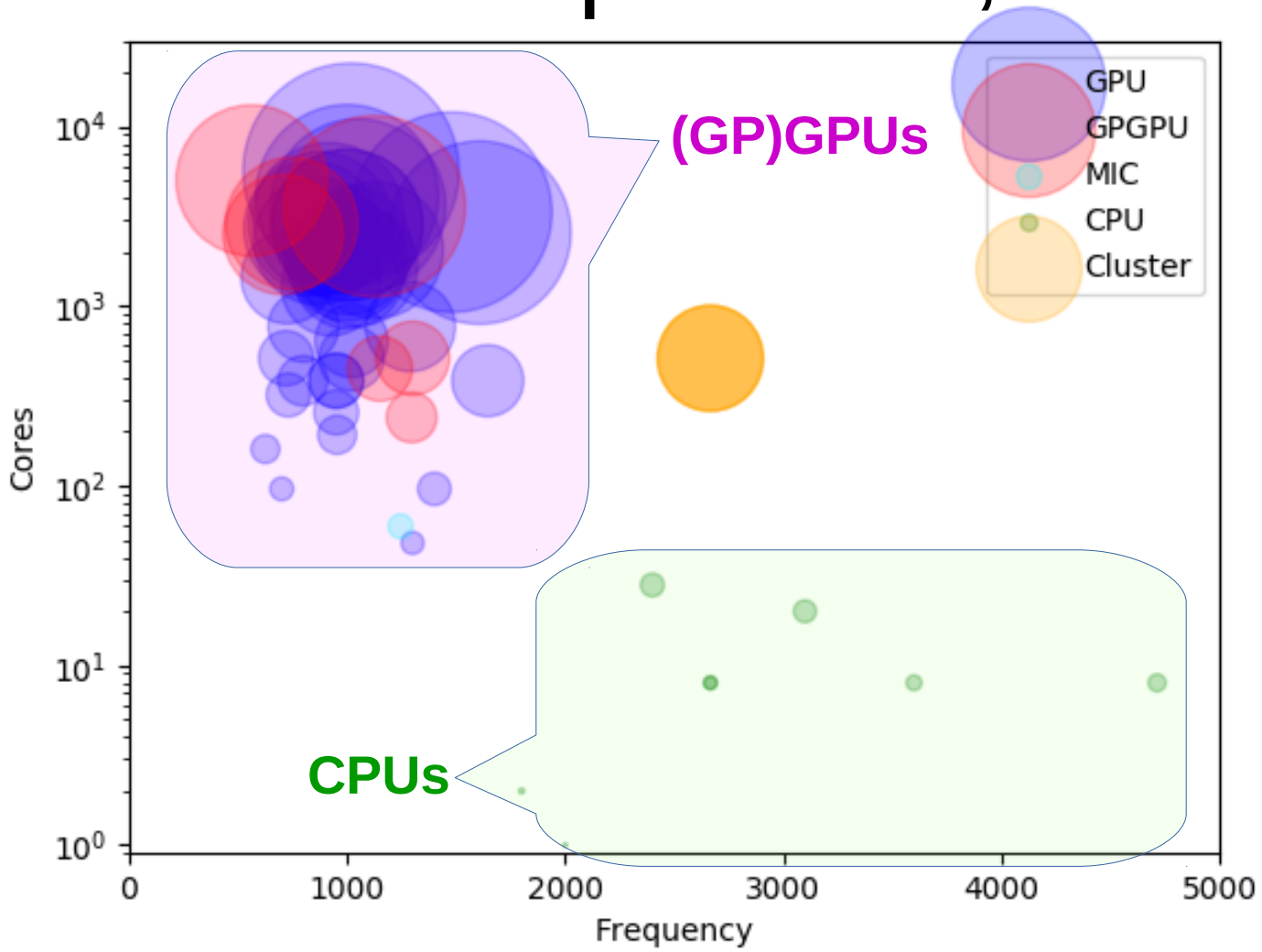
GPU AMD Gamer & al : 18

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

Emmanuel Quemener

October 10, 2018

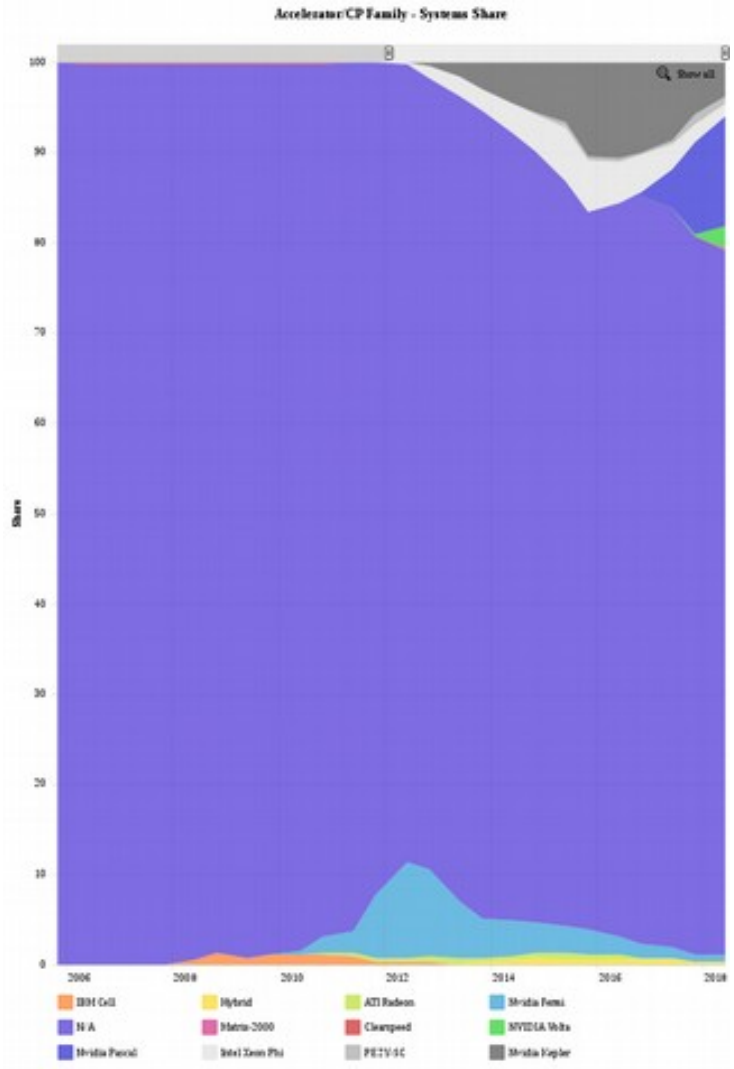
How to represent these testbenchs? Question of frequencies, cores, ...



On TechBenchs in CBP: SIDUS
Never installed, just started !

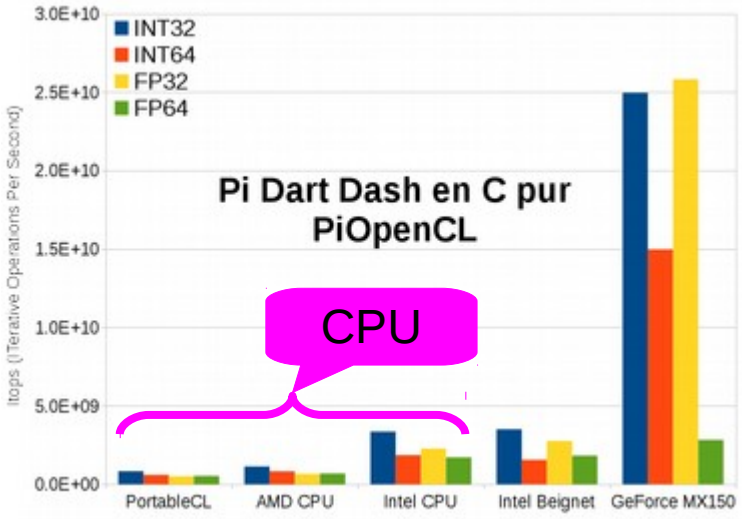
- What ?
  - Deploying a system simply on a set of machines
- Why ?
  - To perform unicity of configurations
  - To Limit the footage of systems on drives
- For who ?
  - Researchers, teachers, students, engineers, ...
- When & Where ?
  - Centre Blaise Pascal : since 2010, near from 200 machines
  - PSMN : since 2011, more than 500 nodes (their proper instance)
  - Laboratories : UMPA, LBMC, IGFL, CRAL, ...
- How ?
  - To use a network share folder
  - To divert a hack used in liveCD since decades

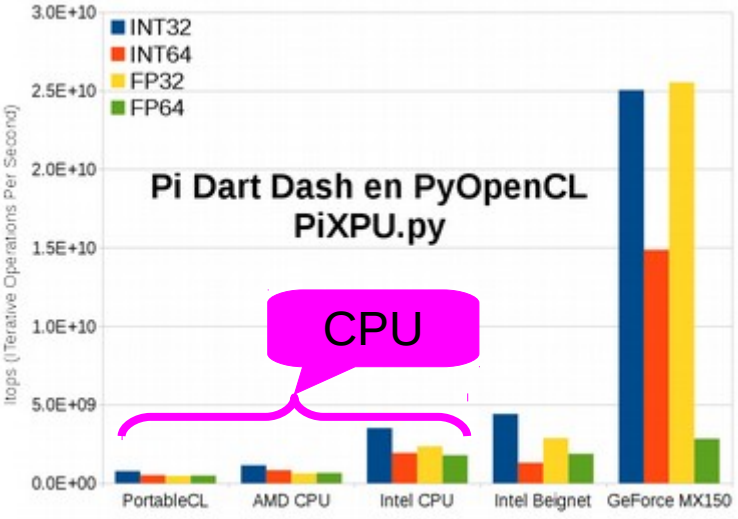
Where are we ?
TOP 500 & the accelerators

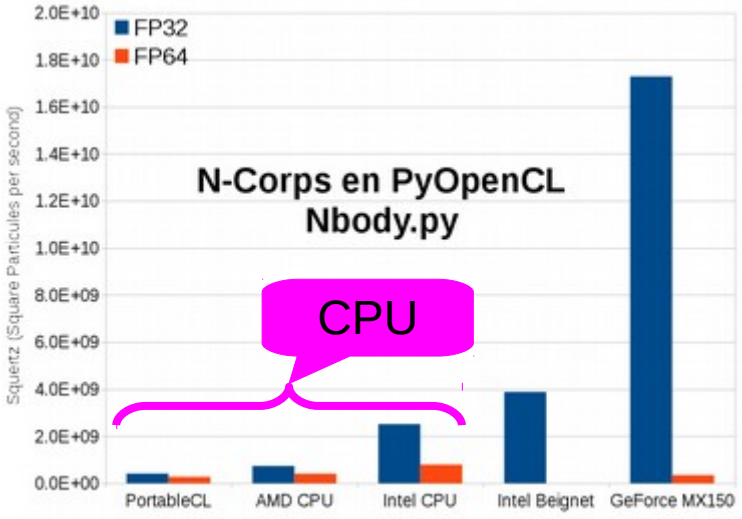


- Where : on Top 500
- When : Jun, 2018
- How much :
  - ~1/4 with accelerators :
    - MIC (Intel & copied ones)
    - GPGPU (Nvidia, AMD)
  - 7/10 in Top 10

Where are we ? What to expect ? On a laptop, CPU & GPU benchs…

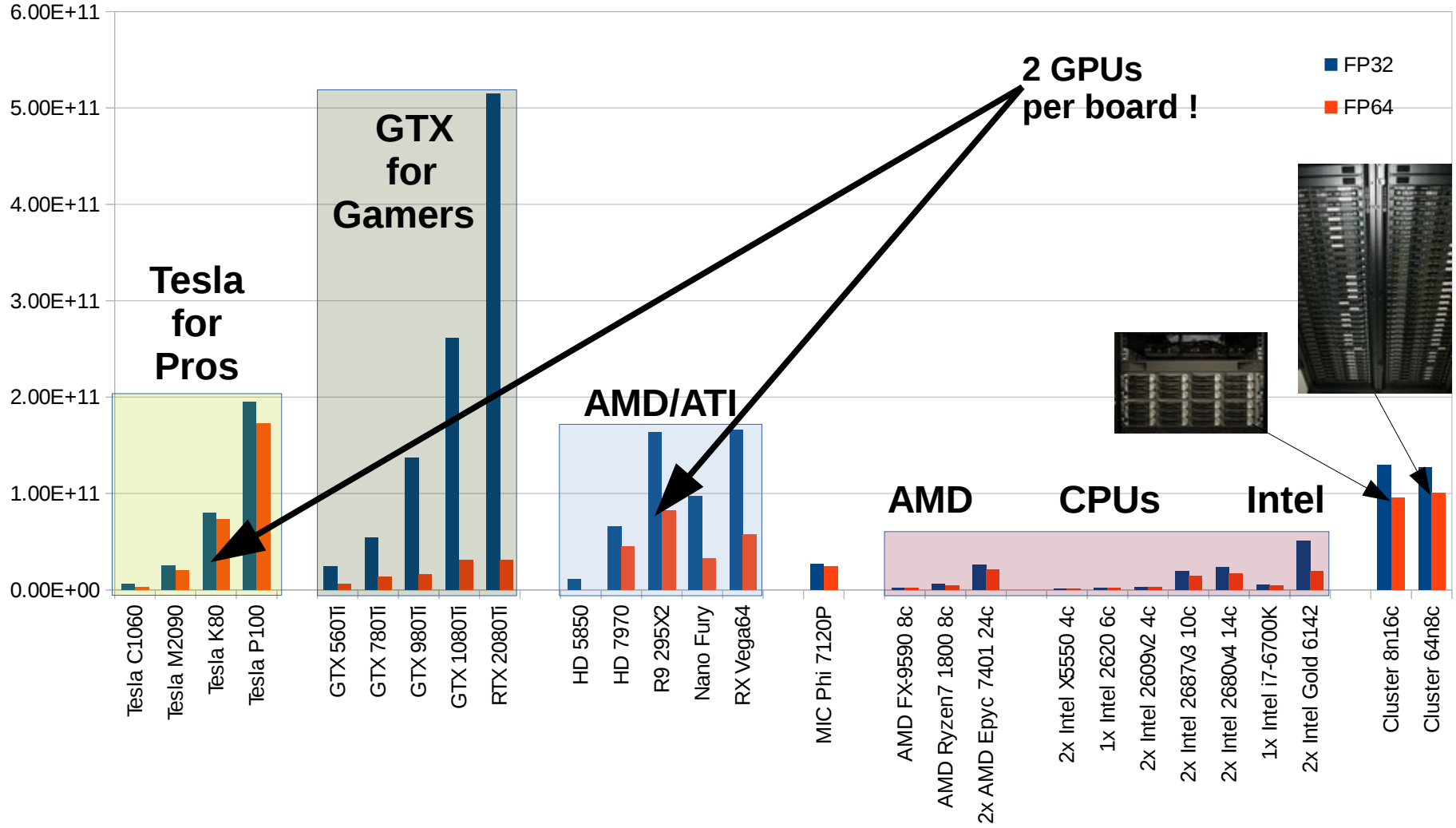




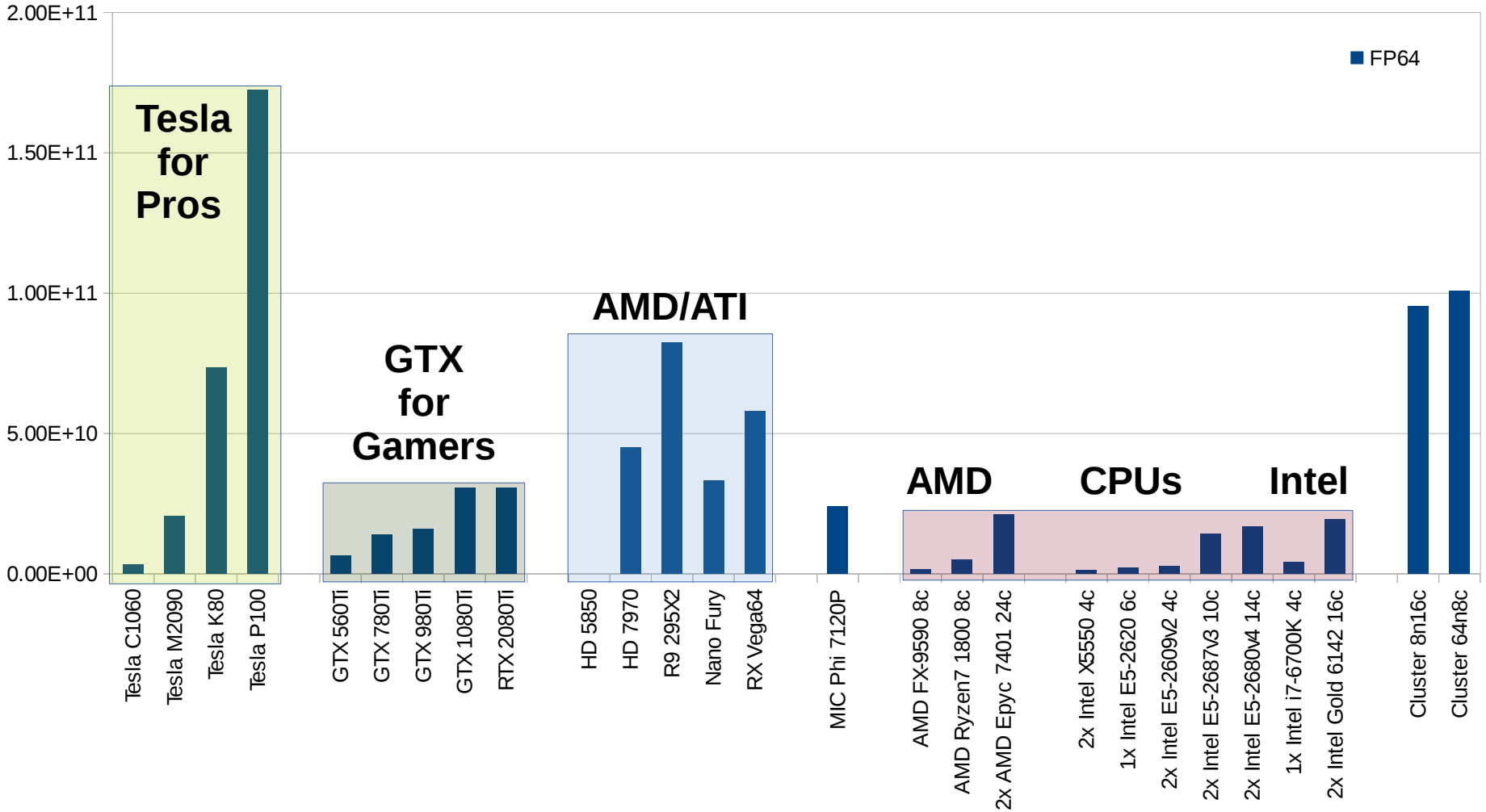


- A « Pi Dart Dash »
  - In C/OpenCL
  - In Python OpenCL
- A naive « N-Body »

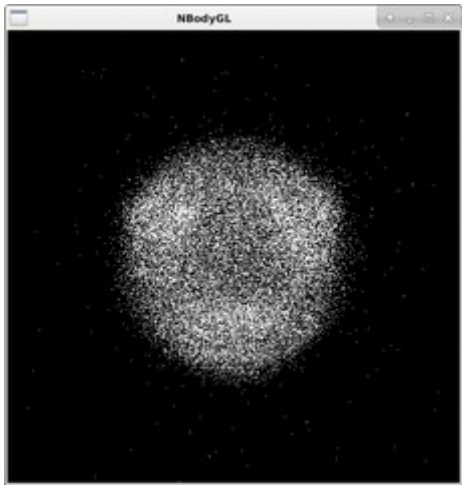
Raw performances on generations... A « Pi Dart Dash » in OpenCL



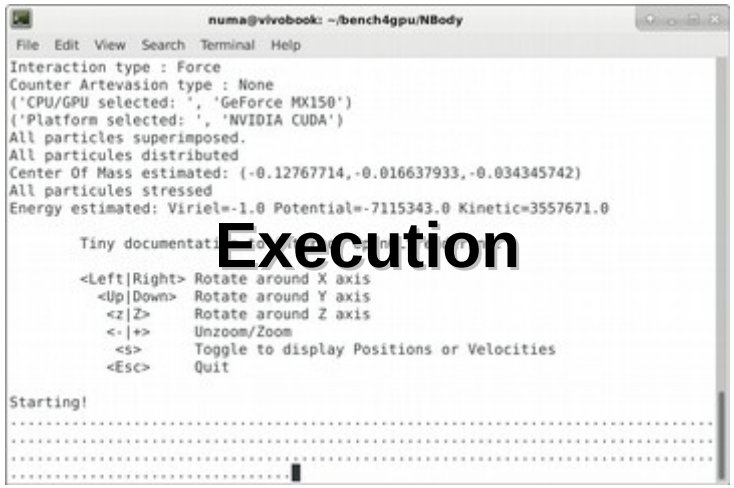
Raw performances in FP64
The same « Pi Dart Dash »



For a more « physical » code Nbody.py in OpenCL/OpenGL



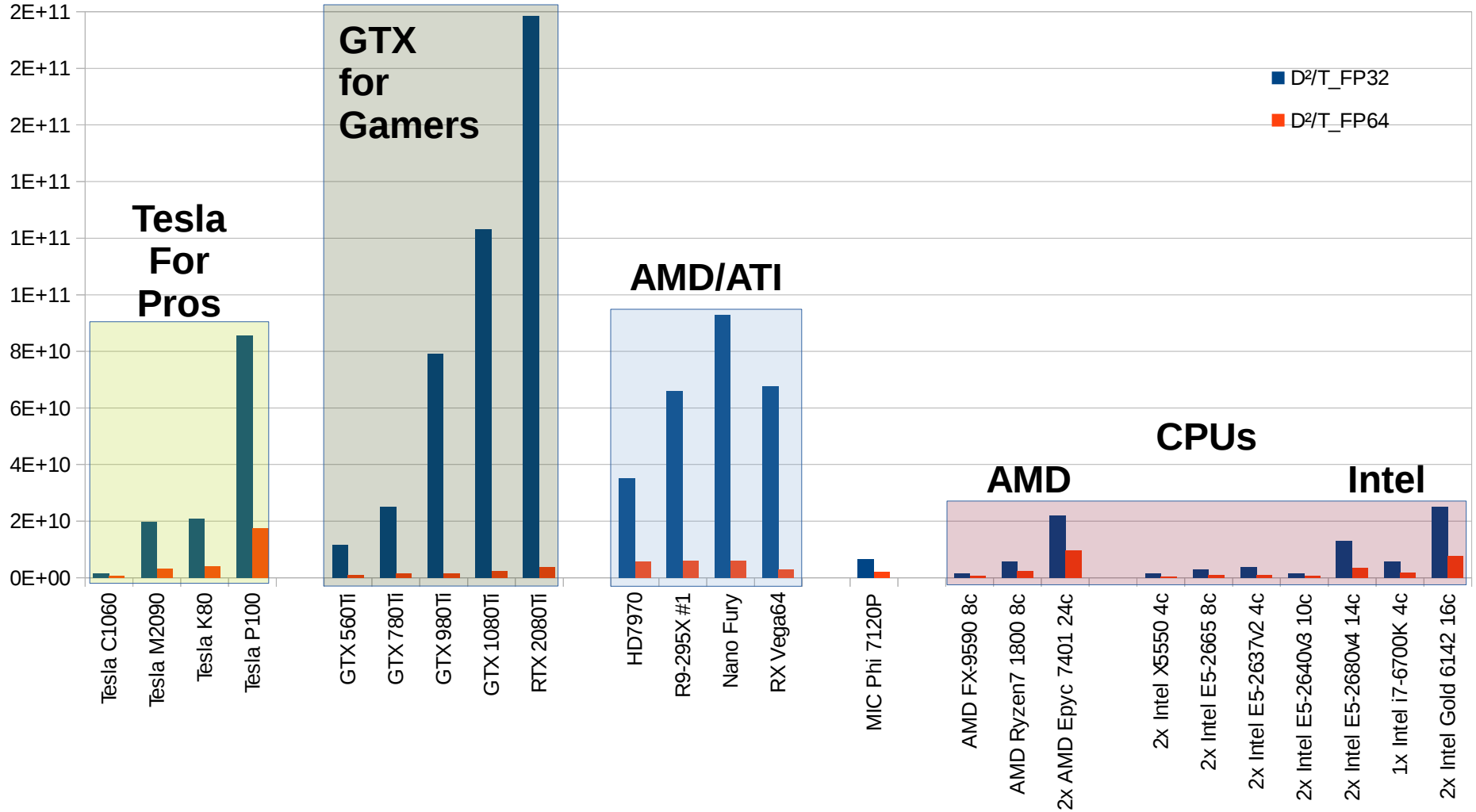
Statistics



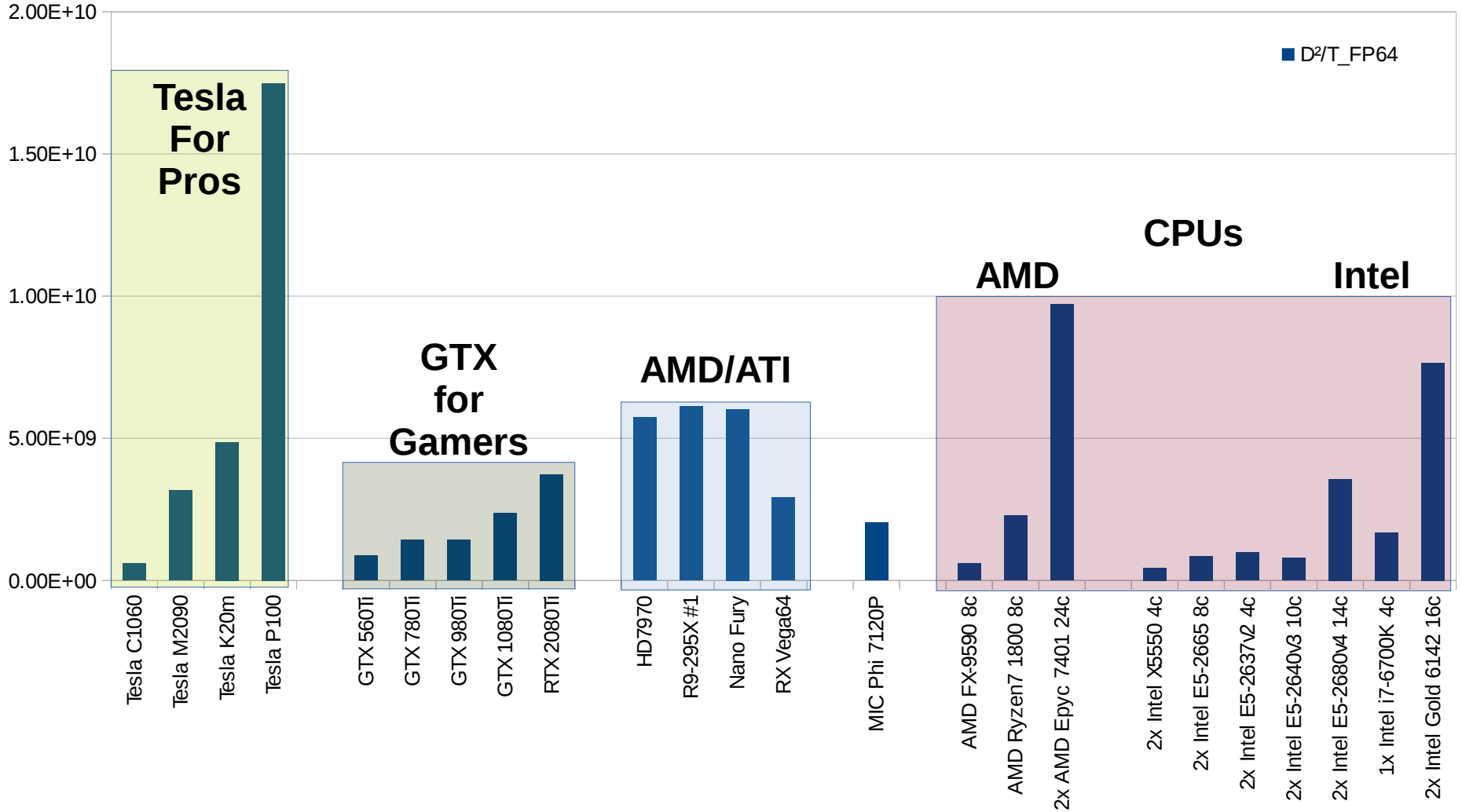
Execution

- Deliberately simple
- Euler implicit implementation
- 32768 particules
  - How many distances computed each step ?

For a more « physical » code « N-Body » newtonian code



In double precision, another lecture GPGPU, the Best. CPU not so bad !



Xeon Phi is mentioned
A stillborn, but instructive

- 3 ways of programming
  - Intel compiler, cross-compiling, execution on embedded micro-system
  - Intel compiler, OpenMP in « offload » mode, transparent execution
  - Intel Implementation of OpenCL for MIC
- What is offload in OpenMP ?

Classical OpenMP on independant tasks

```
#pragma omp parallel for
  for (int i=0 ; i<process; i++) {
inside[i]=MainLoopGlobal(iterations/process,seed_w+i,seed_z+i);
  }
```

Offload call on Xeon Phi MIC

```
#pragma omp target device(0)
#pragma omp teams num_teams(60) thread_limit(4)
#pragma omp distribute
  for (int i=0 ; i<process; i++) {
inside[i]=MainLoopGlobal(iterations/process,seed_w+i,seed_z+i);
  }
```

And in C/OpenCL implementation No Really simple… But wait !

Select the platform & device

- err = clGetPlatformIDs(0, NULL, &platformCount);
- platforms = (cl_platform_id*) malloc(sizeof(cl_platform_id) * platformCount);
- err = clGetPlatformIDs(platformCount, platforms, NULL);
- err = clGetDeviceIDs(platforms[MyPlatform], CL_DEVICE_TYPE_ALL, 0, NULL, &deviceCount);
- devices = (cl_device_id*) malloc(sizeof(cl_device_id) * deviceCount);
- err = clGetDeviceIDs(platforms[MyPlatform], CL_DEVICE_TYPE_ALL, deviceCount, devices, NULL);
- cl_context GPUContext = clCreateContext(props, 1, &devices[MyDevice], NULL, NULL, &err);
- cl_command_queue cqCommandQueue = clCreateCommandQueue(GPUContext,devices[MyDevice], 0, &err);
- cl_mem GPUInside = clCreateBuffer(GPUContext, CL_MEM_WRITE_ONLY, sizeof(uint64_t) * ParallelRate, NULL, NULL);
- cl_program OpenCLProgram = clCreateProgramWithSource(GPUContext, 130 ,OpenCLSource,NULL,NULL);
- clBuildProgram(OpenCLProgram, 0, NULL, NULL, NULL, NULL);
- cl_kernel OpenCLMainLoopGlobal = clCreateKernel(OpenCLProgram, "MainLoopGlobal", NULL);

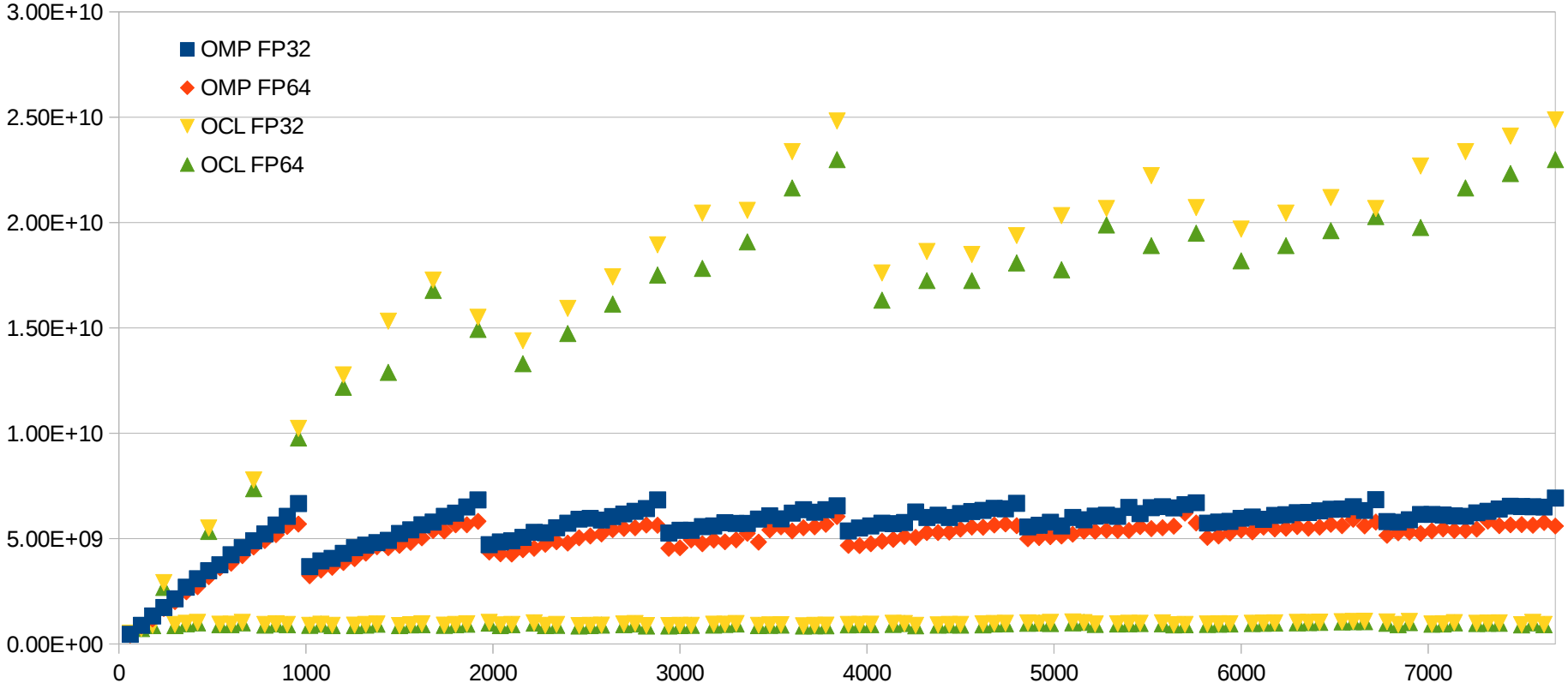
Set attributes to OpenCL kernel

- clSetKernelArg(OpenCLMainLoopGlobal, 0, sizeof(cl_mem),&GPUInside);
- clSetKernelArg(OpenCLMainLoopGlobal, 1, sizeof(uint64_t),&IterationsEach);
- clSetKernelArg(OpenCLMainLoopGlobal, 2, sizeof(uint32_t),&seed_w);
- clSetKernelArg(OpenCLMainLoopGlobal, 3, sizeof(uint32_t),&seed_z);
- clSetKernelArg(OpenCLMainLoopGlobal, 4, sizeof(uint32_t),&MyType);
- size_t WorkSize[1] = {ParallelRate}; // one dimensional Range

- clEnqueueNDRangeKernel(cqCommandQueue, OpenCLMainLoopGlobal, 1, NULL, WorkSize, NULL, 0, NULL, NULL);
- clEnqueueReadBuffer(cqCommandQueue, GPUInside, CL_TRUE, 0, ParallelRate * sizeof(uint64_t), HostInside, 0, NULL, NULL);

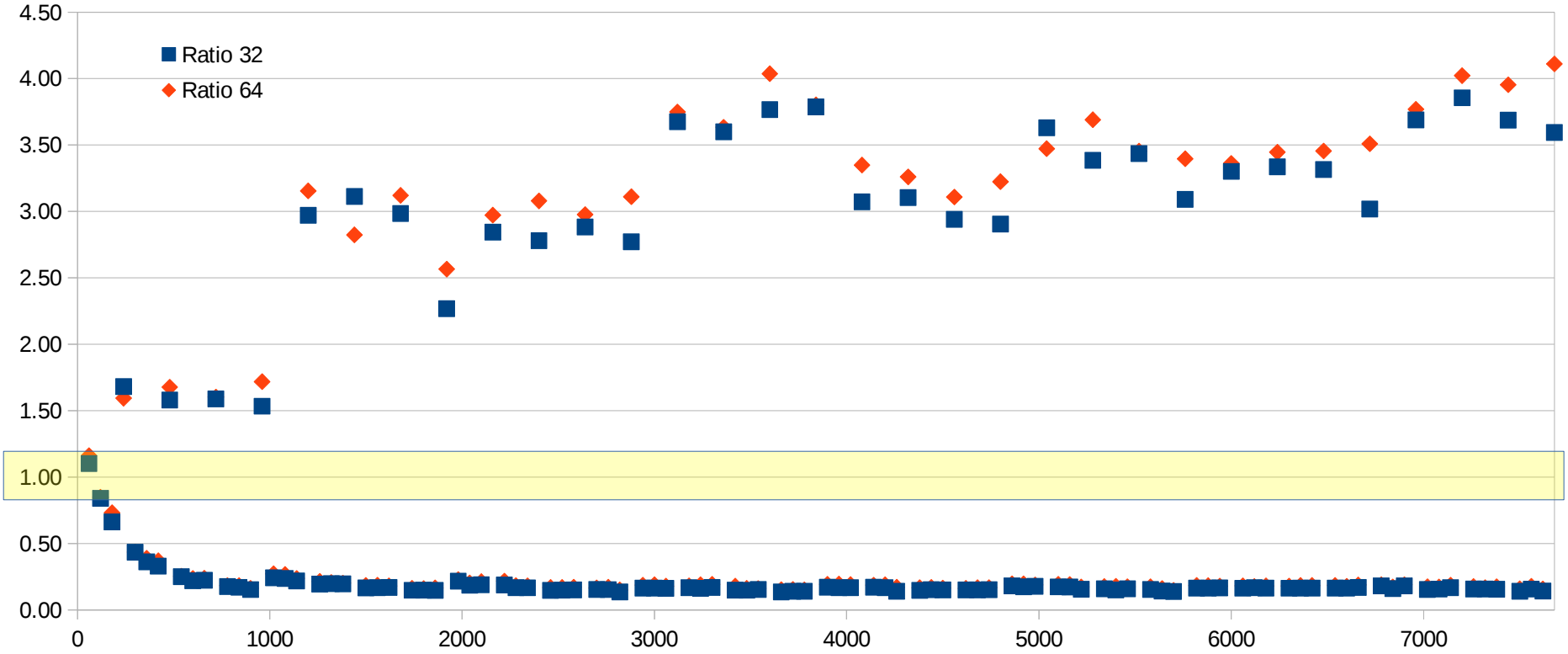
And what about the performances on the Xeon Phi with 60 cores

- Performance in itops from 60 to 7680, step 60
  - OpenMP Offload : Parallel Rate better for 32x the number of Cores, optimum at PR=7680
  - OpenCL : better than OpenMP, only when PR=4x the number of cores, optimum at PR=3840



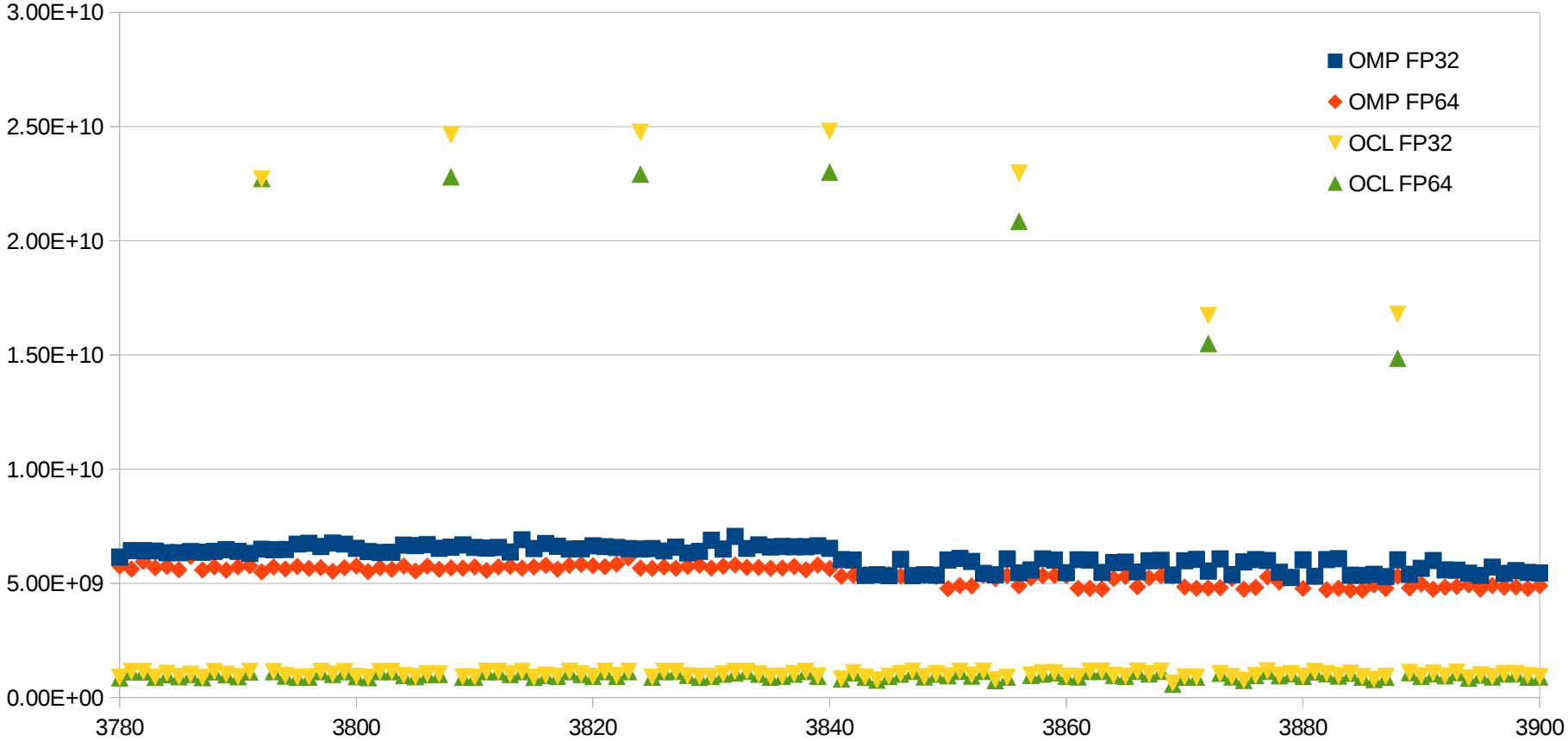
Performances for Xeon Phi Ratio OpenCL/OpenMP

- Speed-up OpenCL > 2.5 for multiples of 240 as PR
- Speed-down OpenCL for all other numbers of PR



Performances around local max

- A period of 16 on Parallel Rate between max performances in OpenCL
- A performance divided by 40 between max and min as Parallel Rate for OpenCL
- A stability of performance for OpenMP



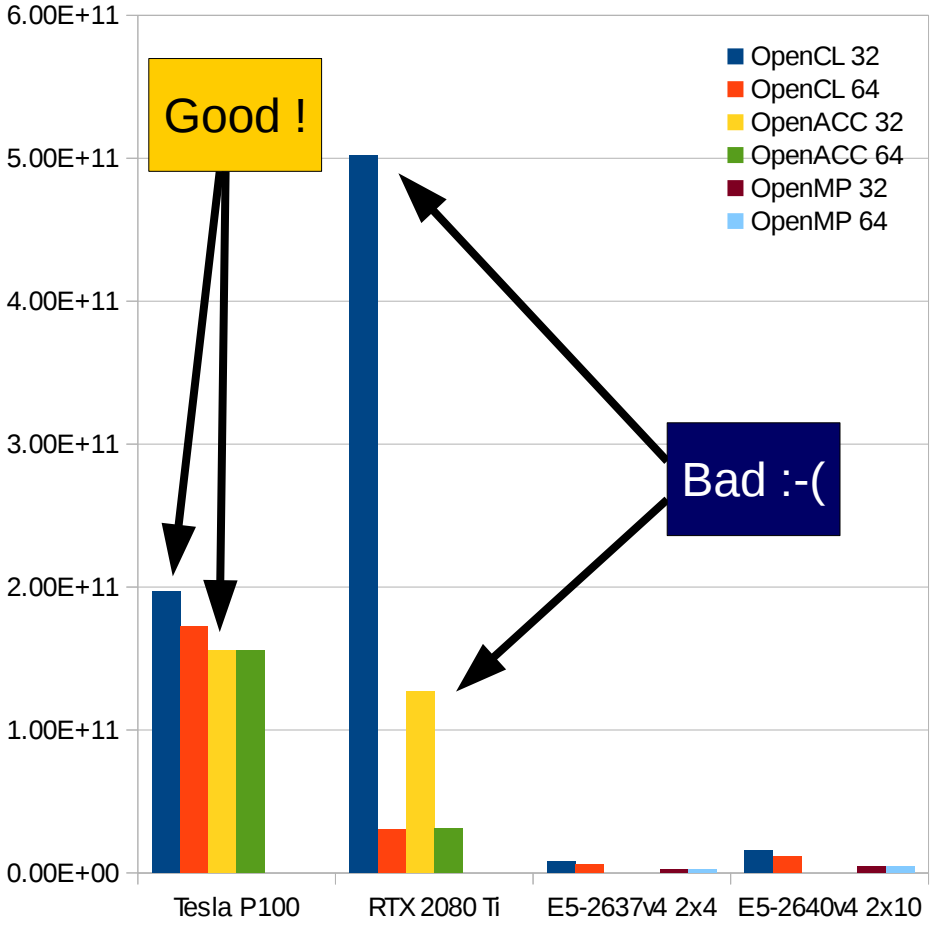
What OpenACC promises ?

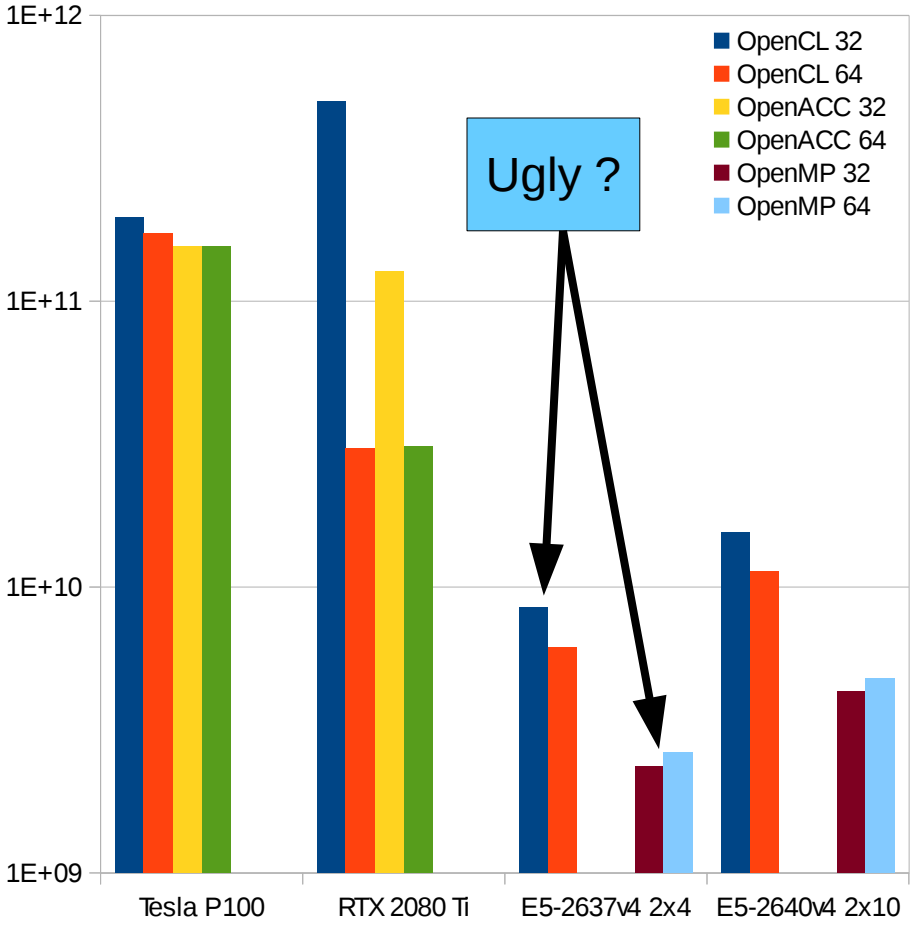
- The same thing as Intel promised in OpenMP offload ?
- From presentation : https://www.olcf.ornl.gov/wp-content/uploads/2012/08/OpenACC.pdf
  - Parallel Construct
    - #pragma acc parallel [clause [[,] clause]...] new-line
  - Data Constructs
    - #pragma acc data [clause [[,] clause]...] new-line
  - Loop Constructs
    - #pragma acc loop [clause [[,] clause]...]new-line
- Why not... But can we test it ?
  - Yes ! From Portland, a complete implementation of OpenACC on Nvidia

OpenACC on « Pi Dart Dash » Very simple implementation...

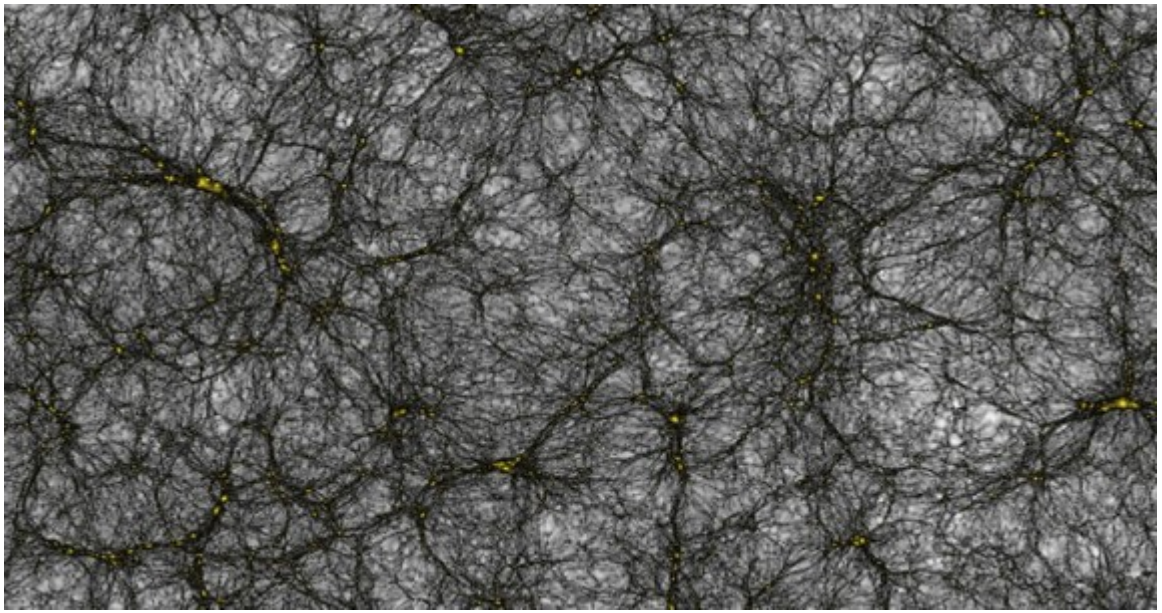
- Previous core routine :
  - #pragma acc routine
  - LENGTH MainLoopGlobal(LENGTH iterations,unsigned int seed_w,unsigned int seed_z)
- Previous distribution loop
  - #pragma omp parallel for shared(ParallelRate,inside)
  - #pragma acc kernels loop
  - for (int i=0 ; i<ParallelRate; i++) {
  - inside[i]=MainLoopGlobal(IterationsEach,seed_w+i,seed_z+i); }

Open ACC/MP in « Pi Dart Dash » Mitigated results…



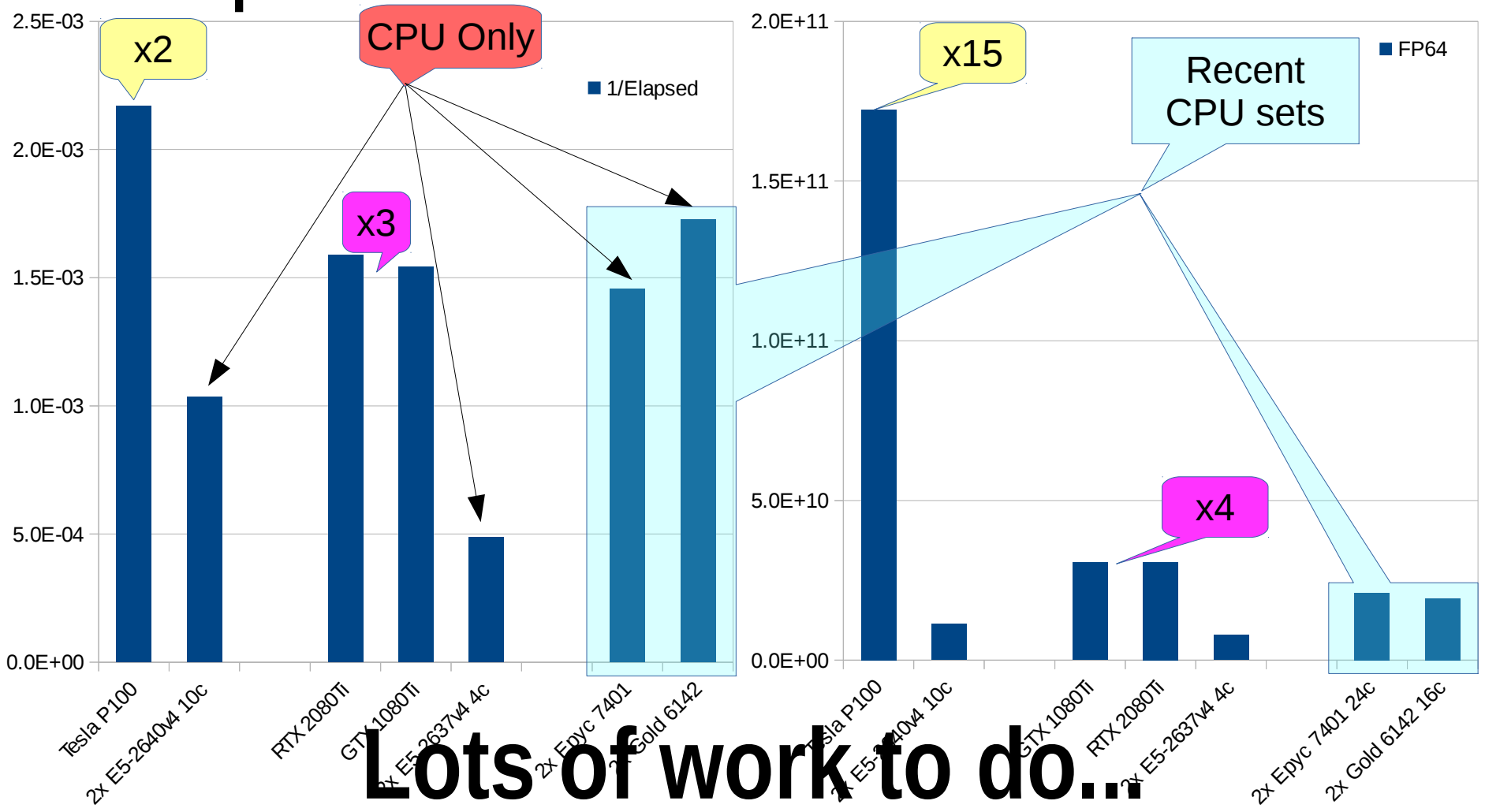


Performance of a GPUfied code PKDGRAV3



- Small hydrid code :
  - MPI
  - OpenMP
  - CUDA (with kernels)
- Compilation :
  - Classical cmake operations BUT old compiler required…
- Execution : 256^3 particules
  - default : to use GPU (be careful in case of multiGPU machine)
  - -cqs 0 : to perform a CPU only simulations

PKDGRAV3 vs PiDartDash in FP64
Comparison between CPU & GPU



Lots of work to do...

How do we go ?

- You have seen :
  - Hybrid approach with primitives : OpenMP, OpenACC
    - The more : easy to use, keeping code legacy
    - The less : licence of compiler & continuity, jailed with Nvidia
  - Kokkos, SyCL approach :
    - The more & the less : C++ & continuity, mostly jailed with Nvidia
- But there is an alternative, the « thinking device » :
  - Using Python as skeleton for calling small & efficient kernels
  - Using OpenCL (or CUDA) for CPU, GPU, GPGPU for kernels
  - Providing « matrix » codes as « inpiring » codes for grain, scalability, etc
  - « Thinking device » demands to fit the parallel rate to device…

Conclusion...

- In all cases : 3 questions to have in mind
  - Do you need FP64 only sometimes but not all the time ?
  - Do your use case compatible with myriad of parallel rate ?
    - Use case : code + data
    - Myriad : number of 10 thousands (optimum around 16 times number of ALU)
  - Do you need less than 16GB of RAM for the core ?
- If yes :
  - (GP)GPU is **a** solution…
  - The Gamer GTX could be much more cheaper !