ETSN 2022

De l’exploration de programmation sur GPU à l’émergence de “Codes Matrices”

“Va toujours par le chemin le plus court, et le plus court est celui qui va selon la nature. Voilà pourquoi il faut agir en tout de la façon la plus naturelle. Une telle ligne de conduite te délivrera de l'emphase, de l'exagération et du style figuré et artificiel…”

Marc Aurèle

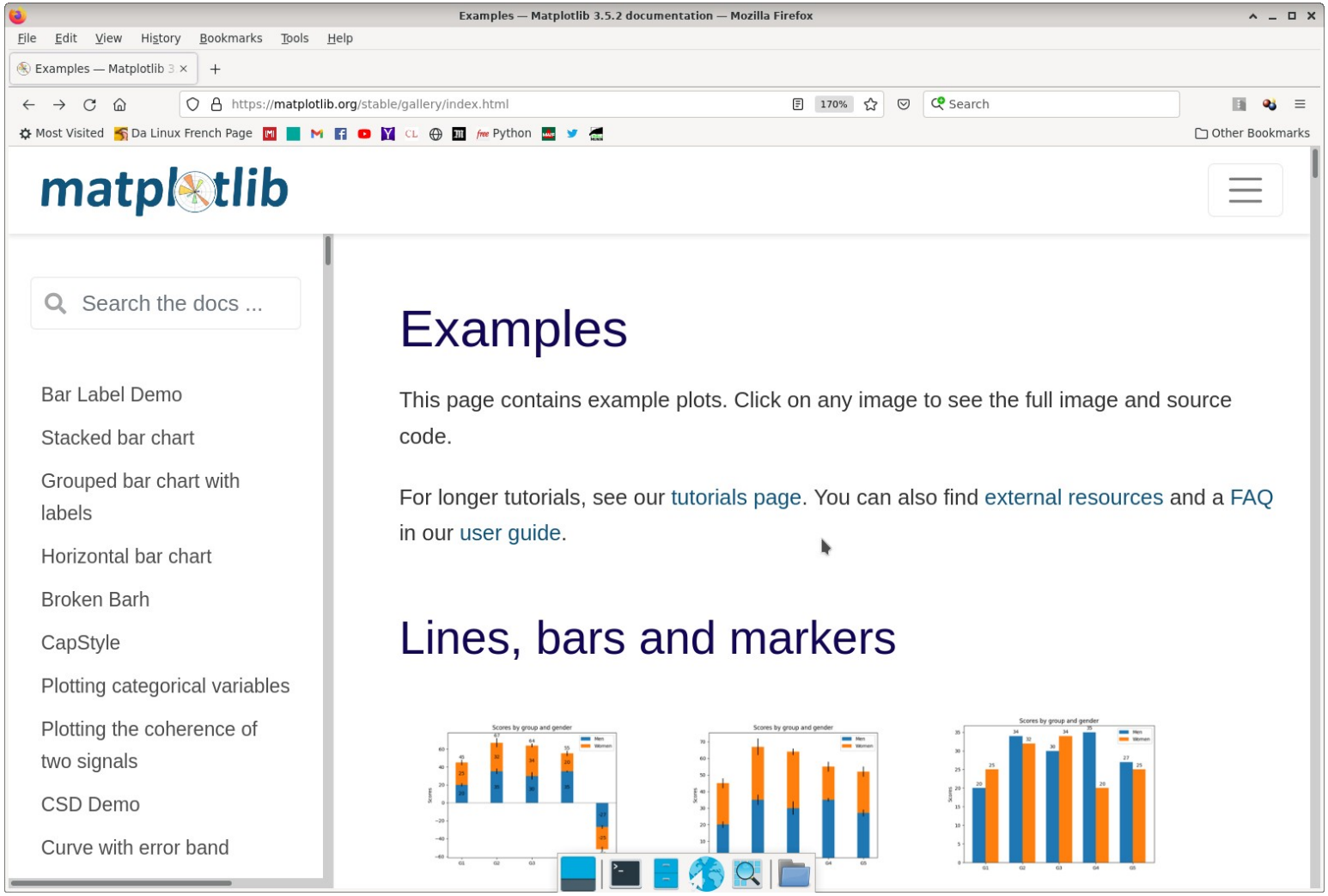
Emmanuel Quémener

Du triptyque du projet…
… au triptyque des coûts

- Pour une gestion de projet simplifiée : les 3 questions...
  - **Où en sommes-nous** ? Etat des lieux : matériel, OS, logiciel, personnes
  - **Où allons-nous** ? Meilleure « performance » mais laquelle ?
  - **Comment y allons-nous** ? Développer, intégrer, adapter, maintenir...
- Face à la gestion des trois coûts :
  - **Coût d'entrée** : appropriation de l'environnement matériel & logiciel
  - **Coût d'exploitation** : MCO et évolution dans cet environnement
  - **Coût de sortie** : sortie ou migration de cet environnement

Tous un peu dans la même galère...

Une approche que j'apprécie : les exemples de Matplotlib...

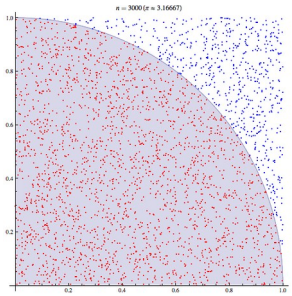


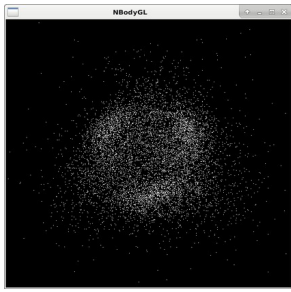
A l'origine, une notion essentielle en sciences : la « référence »

- Histoire personnelle : retour au HPC fin 2009
- Déferlante des (GP)GPU par Nvidia : GTX295 ou C1060
- Code de référence : LinPack du Top 500
  - Exercice personnel : portage du hpl de BLAS à CuBLAS
  - Finalement, plutôt inefficace face aux xBLAS : mais pourquoi ?
- Besoin de revenir aux « fondamentaux »
  - Écrire ses propres programmes exploitant les BLAS
    - Nombreuses implémentations : FBLAS, CBLAS, CuBLAS, clBLAS, gslBLAS…
  - Aborder les problèmes avec deux approches : intégrateur & développeur

Le souci de l'universalité… L'émergence de « codes matrices »

- Pyphi 2011 : modèle d'Ising, transition de phase
  - comparaison C, MPI, OpenCL, CUDA…
- Fin 2012 : code Pi Monte Carlo
  - Gros grain, simple, parallélisable de manière quasi-continue…
  - Charge sur le RNG, nature des variables (INT, FP, 32&64)
  - Implémentations sur toutes les approches parallélisées
- Fin 2014 : code Nbody
  - Grain fin, physique, parallélisable de manière aussi continue
  - Charge sur les calculs, nature des variables (FP32&FP64)





Mais aucune exploitation massive de la mémoire…

Portage « TrouNoir » : fin 2019

- Le Code, le chemin & les gains (temps écoulés) :
  - De Fortran à C en 1997 : x1
  - De C à C/OMP en 2019 : x22 (pour un 28 cœurs)
  - De C/OMP à Python/OpenCL sur CPU : x25 (pour un 8 cœurs)
  - Du CPU au GPU en Python/OpenCL : de x36 à x106
  - De Python/OpenCL à Python/CUDA : de x56 à x156
- Bref…
  - Il n'y a pas que CUDA ou C++ dans la vie… Il y a Python et OpenCL !

En conclusion pour TrouNoir.py...

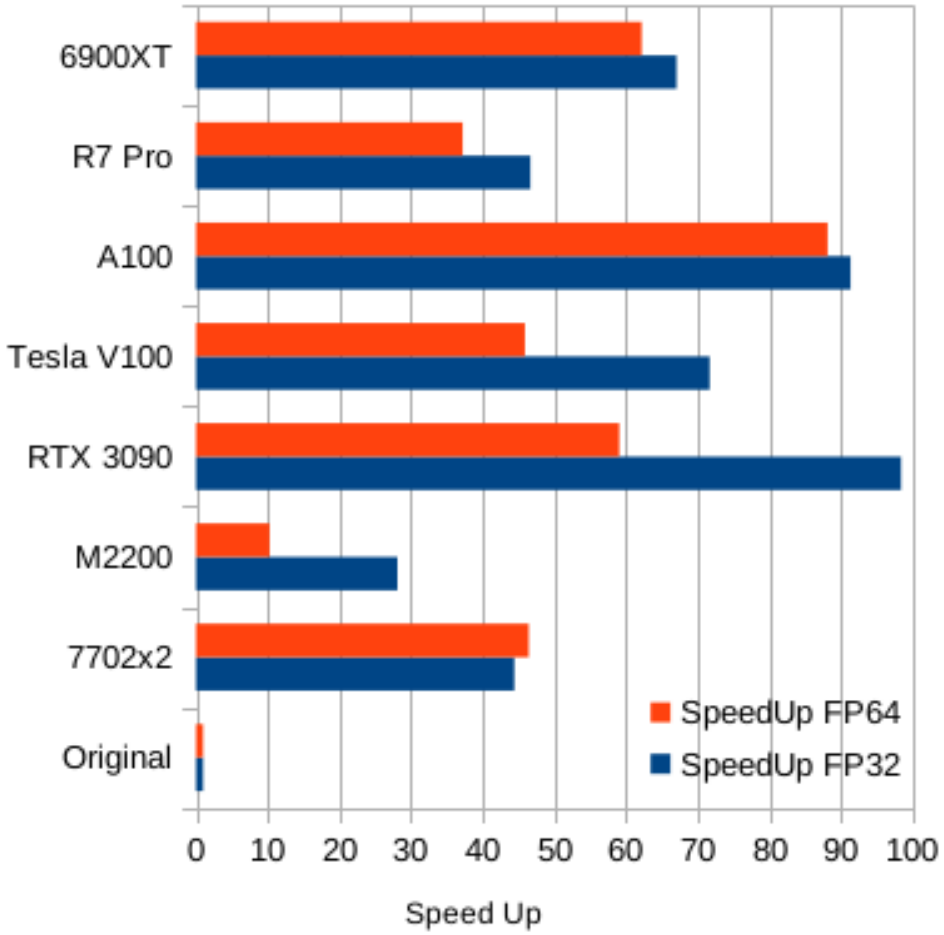
- L'informatique a évolué en 30 : des gains colossaux
  - Principaux gains : FPU (début 80), GPU (début 2010)...
- En 2021, puissance de calcul : GPGPU puis HugeCPU
- Paralléliser ou gépufier un code : une question de temps
  - C'est possible, mais il faut qu'il soit bien « conditionné »
  - En OpenMP, parfois, distribuer des boucles suffit
  - En OpenCL, copier son code C dans du Python et appeler les noyaux
  - En CUDA, s'inspirer de OpenCL et exploiter les « étages » de //isation
- Constater que tout n'est que « cas d'usage »

Vers un portage préliminaire… Application SPH en astrophysique

- Contexte de l'étude :
  - Code issu d'un développement formel sous Mathematica
  - En C++, 2 millisecondes par particule
  - Nécessité de lancer le lancer sur un million de particules
- Objectif : réalisation d'un démonstrateur
  - Portage en Python/OpenCL
  - Comparaison de fichiers de sortie
  - Premier portage : utilisation de « TrouNoir.py » comme « matrice »

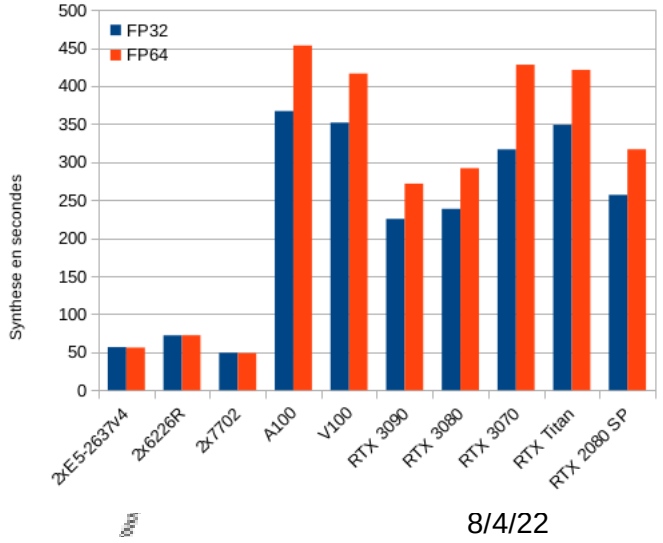
Premiers résultats Encourageants mais déstabilisants

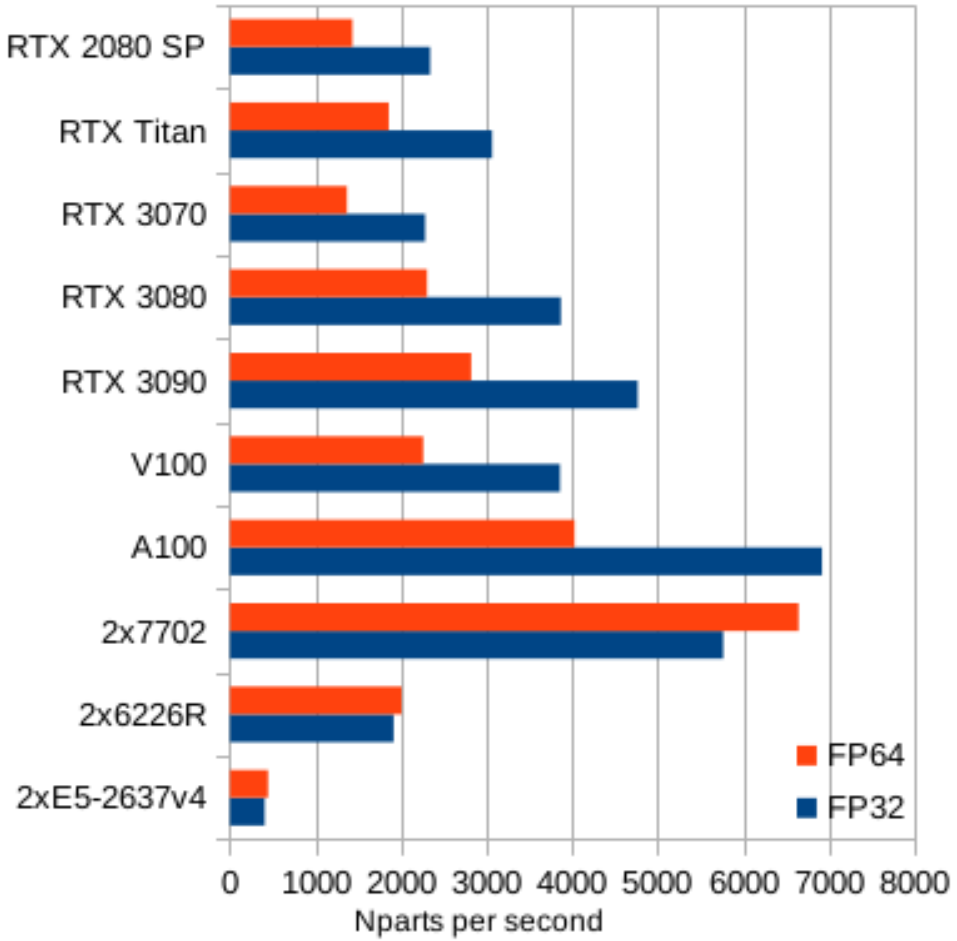
- « Noyaux » OpenCL :
  - du copier/coller
- Essentiel du temps passé :
  - formatage sortie C++...
- Speed Up « confortable »
  - ~45 pour les meilleurs CPU
  - ~90 pour le meilleur GPU
- Proximité FP32 & FP64
  - Code « Memory Bound »



Ordre #1 : des noyaux « énormes »
Les « HugeCPU » impressionnants

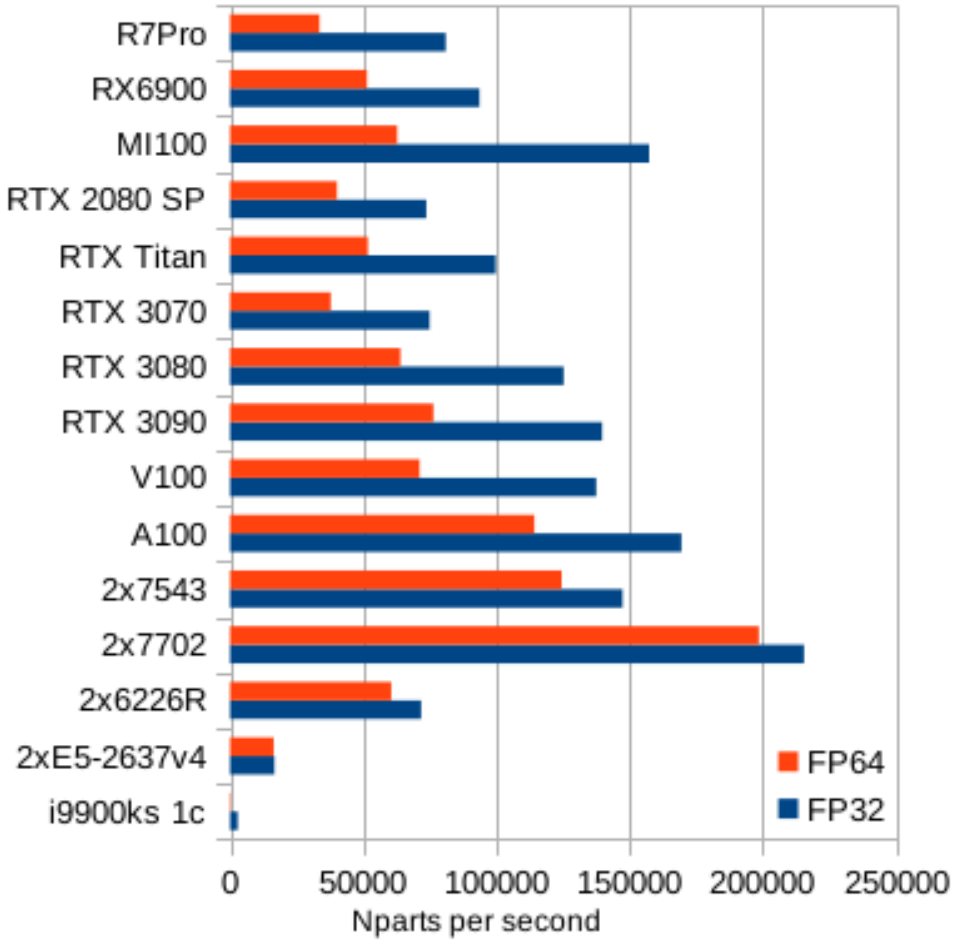
- Ordre #0 : « wc », 26KB
  - 860 lignes, 2057 mots
- Ordre #1 : « wc », 28MB
  - 6581 lignes, 31191 mots
- « Synthèse » préalable...





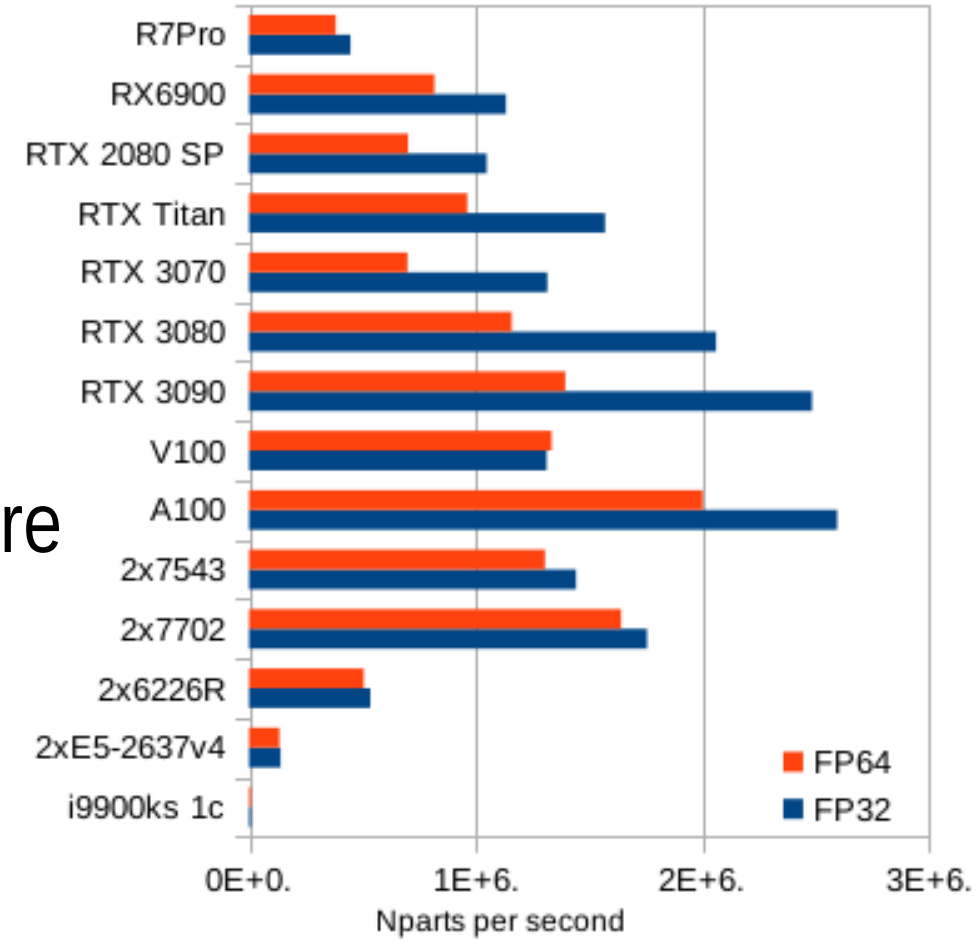
Ordre #2 : des noyaux « purgés » Les « HugeCPU » l'emportent...

- Ordre #1 : « wc » de 372KB
  - 5020 lignes, 15423 mots
- Ordre #2 : « wc » de 3.7MB
  - 6132 lignes, 20054 mots
- « Synthèse » ~ seconde
- Résultats
  - HugeCPU en tête
  - HugeGPGPU sur le podium

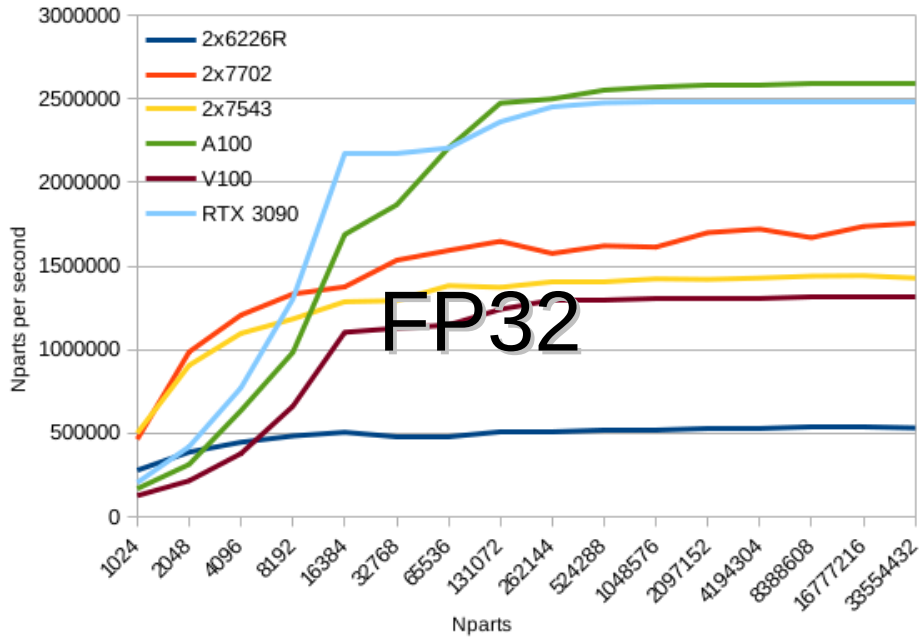


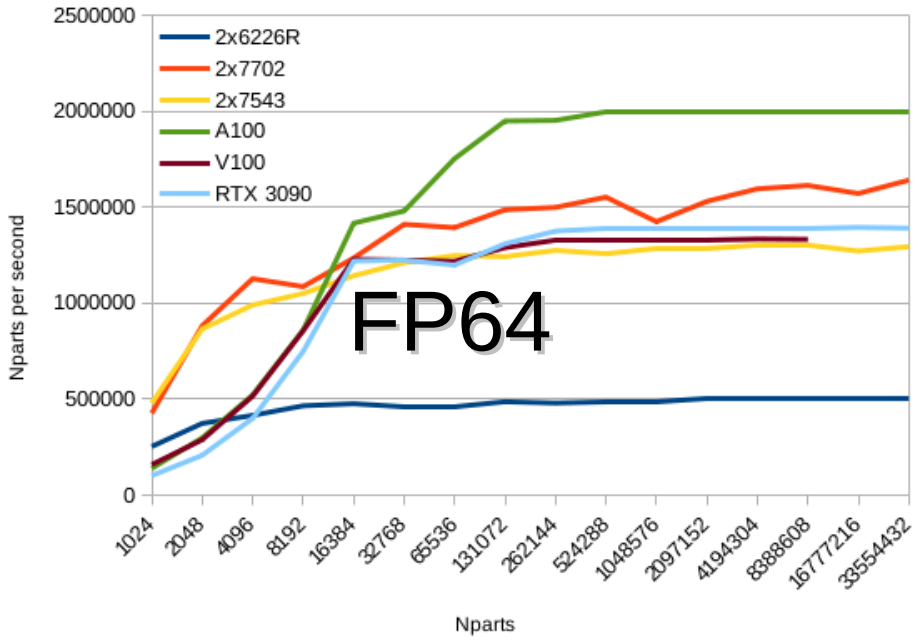
Ordre #2 : nouvelle implémentation
Les (GP)GPU d'une courte tête...

- Un gain de 10 au total
  - L'algorithme a du bon
- La Nvidia A100 en tête
  - Les Ampère Gamer derrière
- Les HugeGPU juste derrière
  - Milan bien meilleur que Rome
- Confirmation...



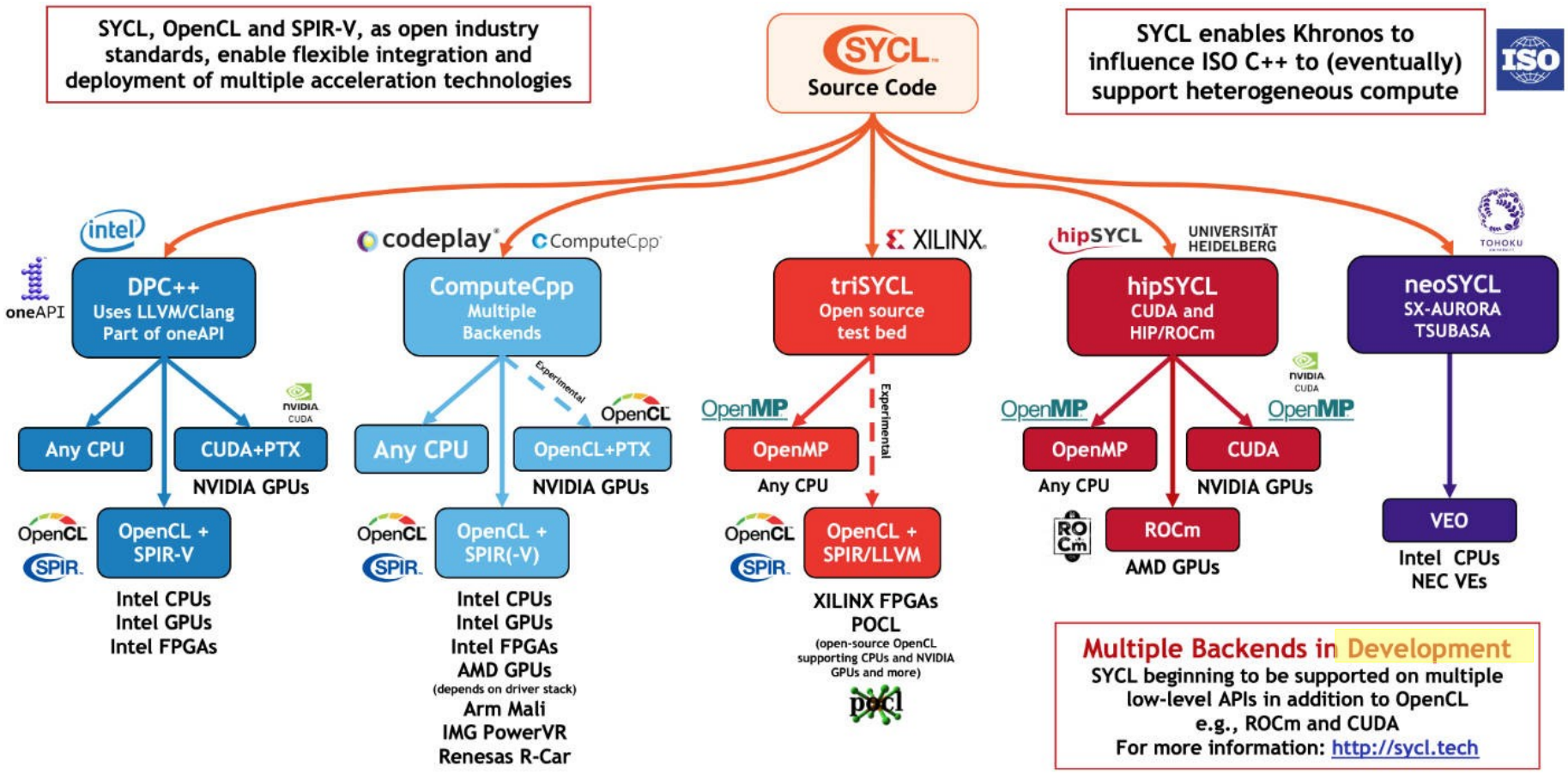
Conclusion du démonstrateur « Montée en charge » : le critère...





- Distribution optimale : > à 200000...
- Equipe CRAL : préférence pour SyCL
  - Un doctorant « démerdard » comme référence interne...

SyCL : « l'avenir » de OpenCL ?



- En fait, TOUT est en développement...

SyCL Intel sur GitHub : Juste pour donner une idée...

Build DPC++ toolchain with support for NVIDIA CUDA

There is experimental support for DPC++ for CUDA devices.

To enable support for CUDA devices, follow the instructions for the Linux or Windows DPC++ toolchain, but add the `--cuda` flag to `configure.py`. Note, the CUDA backend has experimental Windows support, windows subsystem for linux (WSL) is not needed to build and run the CUDA backend.

Enabling this flag requires an installation of CUDA 10.2 on the system, refer to NVIDIA CUDA Installation Guide for Linux or NVIDIA CUDA Installation Guide for Windows

Currently, the only combination tested is Ubuntu 18.04 with CUDA 10.2 using a Titan RTX GPU (SM 71). The CUDA backend should work on Windows or Linux operating systems with any GPU compatible with SM 50 or above. The default SM for the NVIDIA CUDA backend is 5.0. Users can specify lower values, but some features may not be supported. Windows CUDA support is experimental as it is not currently tested on the CI.

Non-standard CUDA location

If the CUDA toolkit is installed in a non-default location on your system, two considerations must be made.

Firstly, **do not** add the toolkit to your standard environment variables (`PATH`, `LD_LIBRARY_PATH`), as to do so will create conflicts with OpenCL headers.

Secondly, set the `CUDA_LIB_PATH` environment variable and pass the CMake variable `CUDA_TOOLKIT_ROOT_DIR` as follows:

```
CUDA_LIB_PATH=/path/to/cuda/toolkit/lib64/stubs CC=gcc CXX=g++ python $DPCPP_HOME/llvm/buildbot/configure.py --cuda --cmak

CUDA_LIB_PATH=/path/to/cuda/toolkit/lib64/stubs CC=gcc CXX=g++ python $DPCPP_HOME/llvm/buildbot/compile.py

$DPCPP_HOME/llvm/build/bin/clang++ -std=c++17 -O3 -fsycl -fsycl-targets=nvptx64-nvidia-cuda --cuda-path=/path/to/cuda/tool

LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$DPCPP_HOME/llvm/build/lib ./a.out
```

- Ne fonctionne « que » sur les pilotes Nvidia « externes »

Qu'avez-vous «subi» cette semaine ?

- Une approche « personnelle »
  - *Celle que j'aurai voulu avoir quand j'ai abordé cet apprentissage…*
- Une approche « individuelle »
  - Au CBP, mais aussi sur vos postes personnels (pour d'eux d'entre vous)
- Une approche « contextuelle »
  - Python est « le » langage universel, en enseignement et ailleurs...
- Une approche « progressive »
  - Partis des 2 premiers exemples des documentations PyOpenCL & PyCUDA
- Une approche « portable »
  - Tout développement exécutable sans modification sur toutes machines...

Annuaire de « bonnes pratiques » Pour éviter la déception...

- Le constat de 2021 (différent de 2019) :
  - Le (GP)GPU est performant, mais plus le meilleur (dans tous les cas)...
- Se poser les « bonnes questions » :
  - Combien de tâches indépendantes puis-je lancer ? >100000 ?
  - Quelle est la « charge calculatoire » élémentaire de chacune ? >1000 ?
  - Quelle est « l'empreinte mémoire » de mon exécution ?
  - Quelles sont les communications entre tâches ?
  - Quel est le « grain » de mon application ?
  - Quelle doit être la pérennité de mon code ?
  - Quel doit être l'œcuménisme de mon code ?