

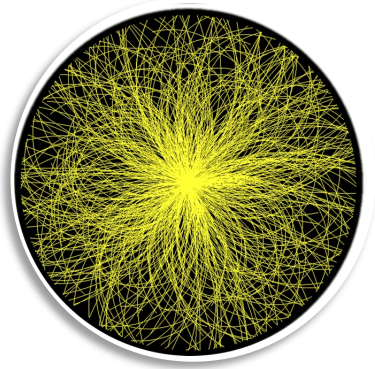

Calcul hétérogène dans les expériences HEP

Adriano Di Florio (CC-IN2P3)

15èmes Journées Informatiques IN2P3/IRFU – 26/09/2024

Intro

CPJ@CC



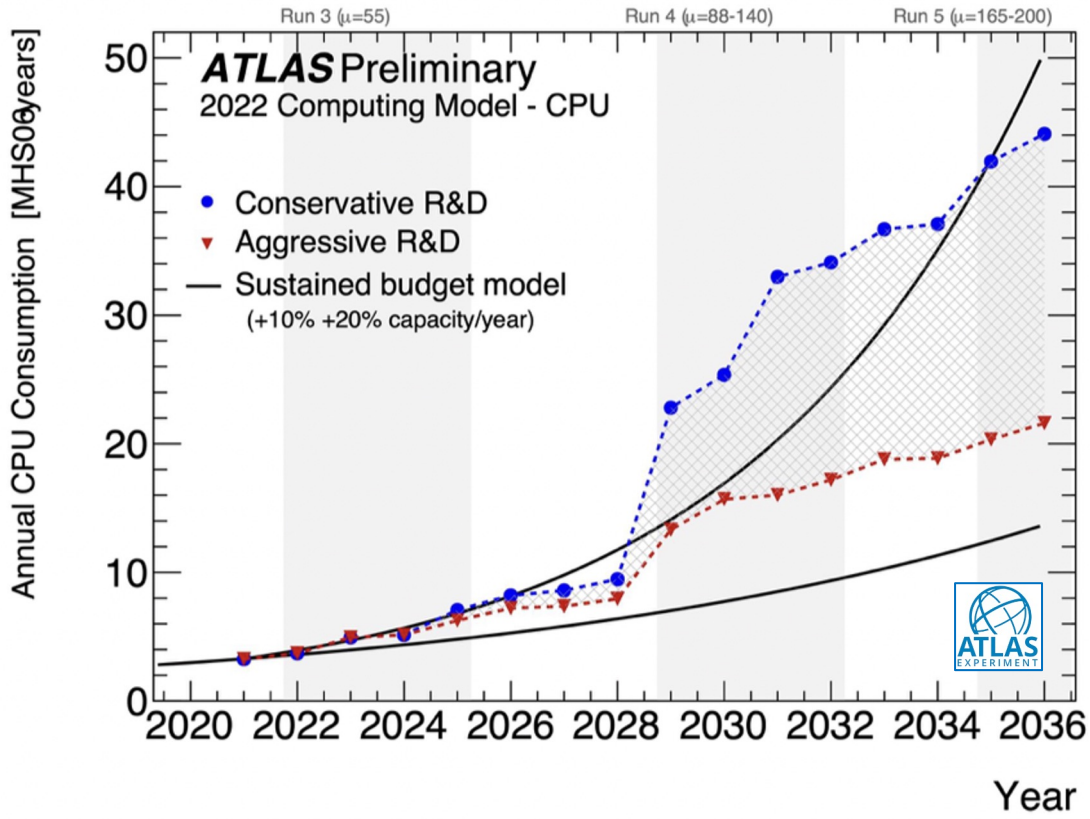
LHC

GPU

Pourquoi?

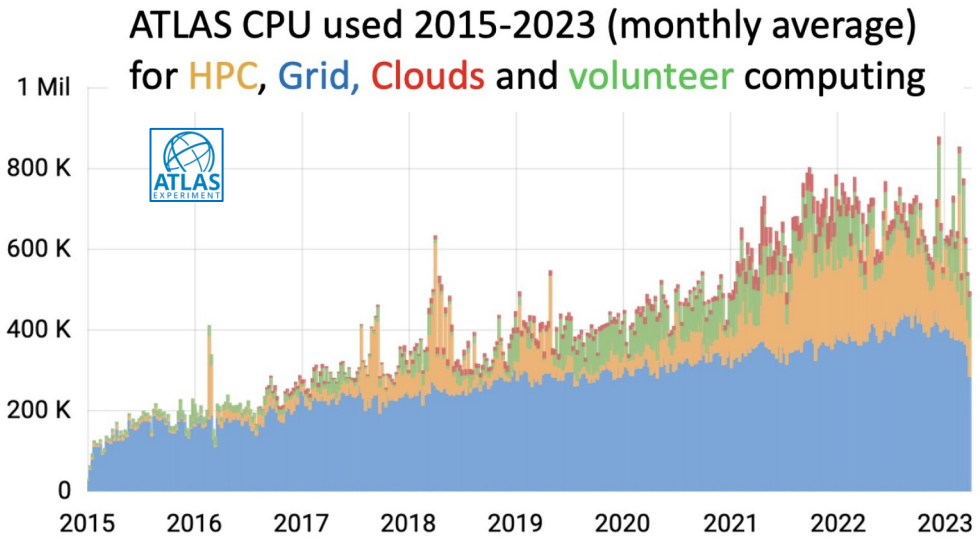
Données

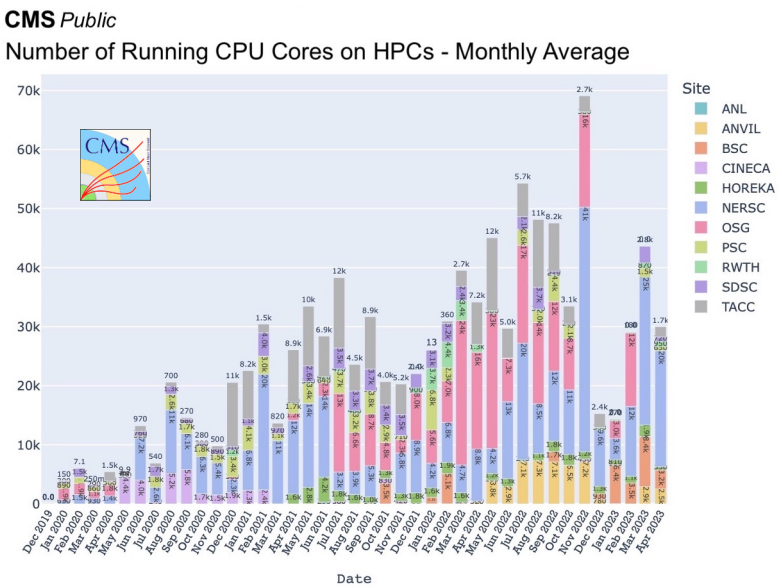
Beaucoup plus de données : doubler la quantité des donnés de la Phase I en un an.



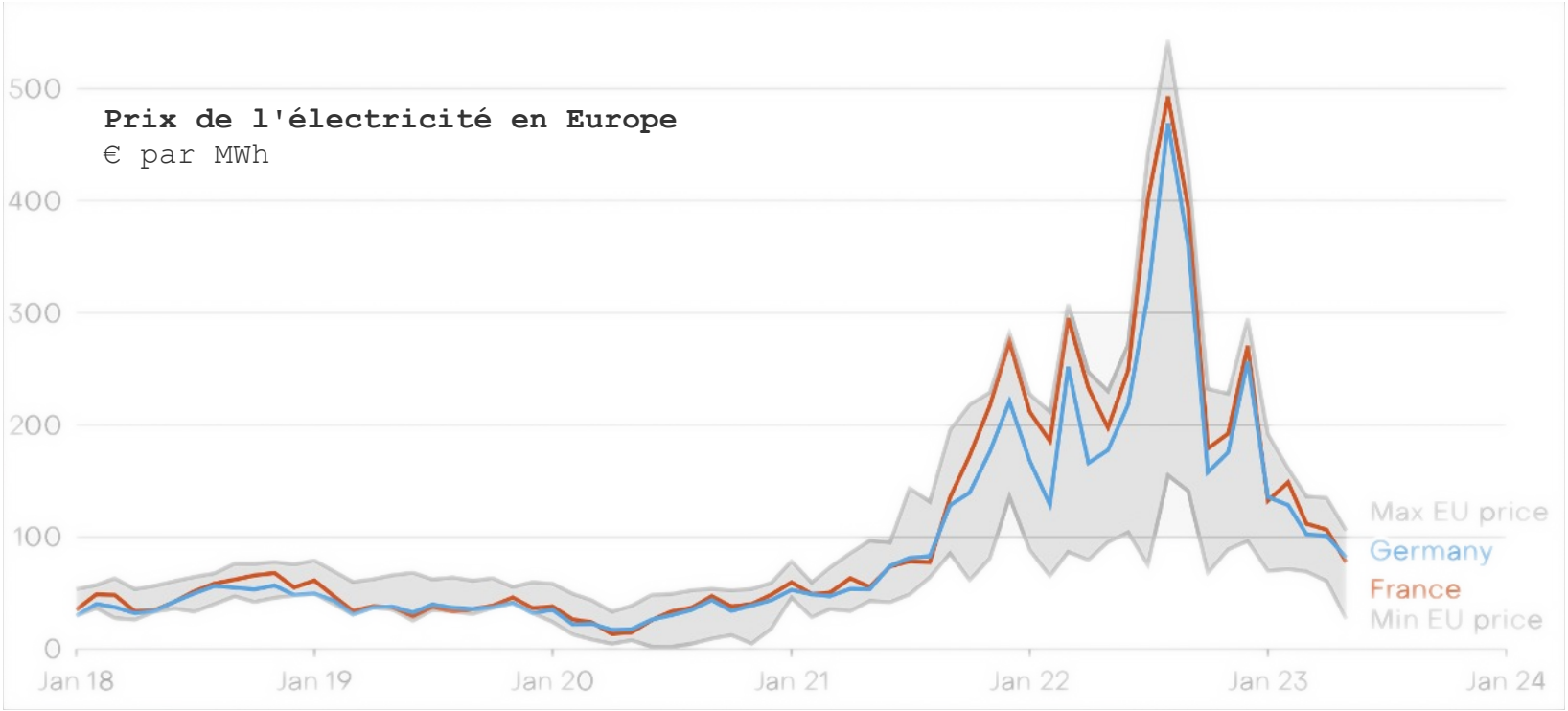
HPCs

« **Plus de HPC** » : de plus en plus visible dans l'infrastructure de calcul LHC. Vaste utilisation des GPU.



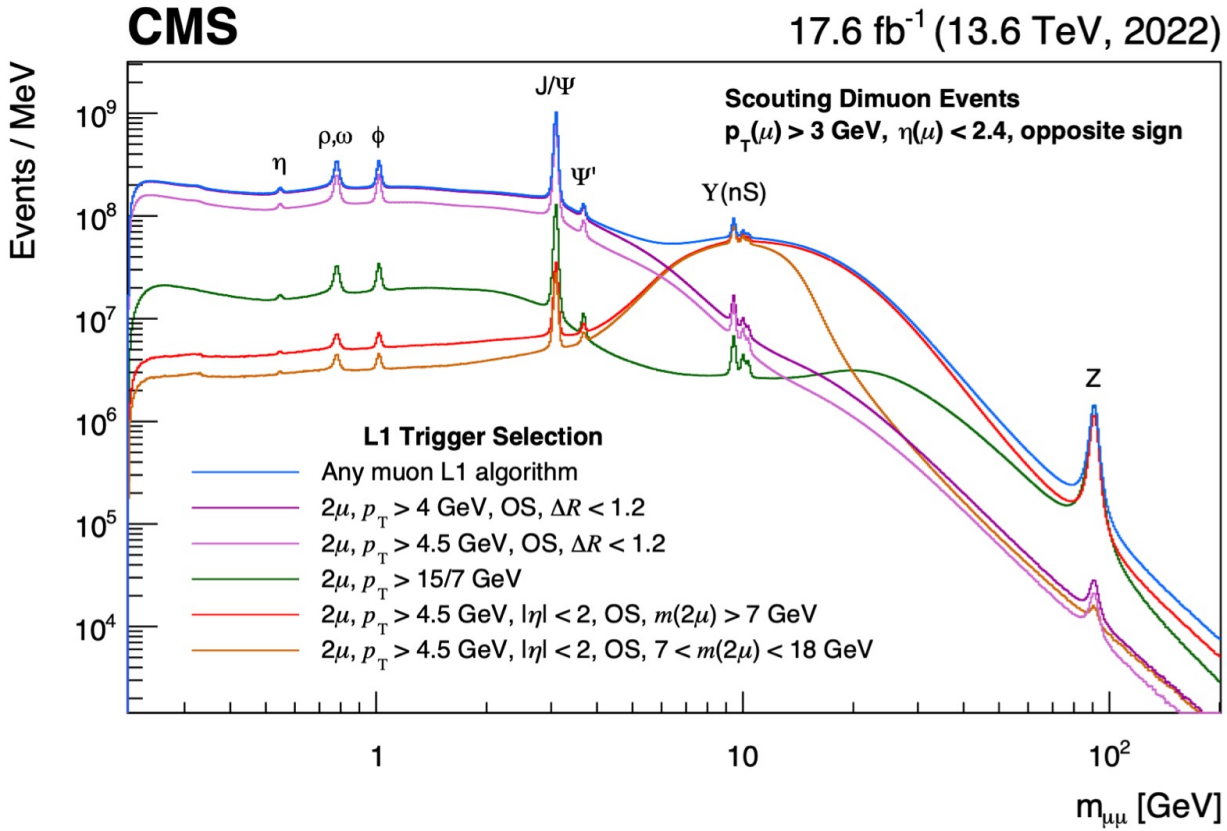


Coûts plus élevés: économiser l'énergie devient plus important que jamais (GPU sont plus «verts»).



Faster == Better

De meilleurs résultats : « plus vite et mieux ». Ex., CMS scouting à 30 kHz.



Calcul hétérogène dans les expériences HEP? Plusieurs questions.

- Comment puis-je accélérer l'exécution du code ?
- Dois-je modifier les structures de données et la gestion de la mémoire ?
- Que se passe-t-il si l'algorithme n'expose pas suffisamment de parallélisme ?
- Et s'il n'y a pas d'algorithme ou de détecteur unique qui soit un point névralgique prenant la majeure partie du temps de reconstruction ?
- Comment intégrer le code dans le cadre ?
- Comment rendre le code portable ?
- Comment s'assurer que les threads de l'unité centrale ne sont pas inactifs en attendant le GPU ?
- Comment équilibrer le nombre et le type de GPU avec la fraction de code GPU dans le logiciel ?
- Comment faire correspondre les besoins en GPU du travail avec les machines disponibles sur la grille ?
- Comment obtenir des résultats reproductibles ?

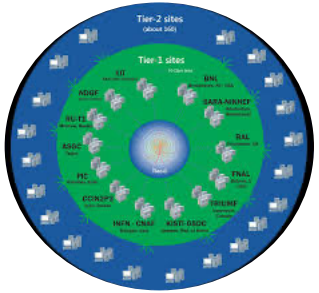
Calcul hétérogène dans les expériences HEP? Plusieurs questions.

- Comment puis-je accélérer l'exécution du code ?
- Dois-je modifier les structures de données et la gestion de la mémoire ?
- Que se passe-t-il si l'algorithme n'expose pas suffisamment de parallélisme ?
- Et s'il n'y a pas d'algorithme ou de détecteur unique qui soit un point névralgique prenant la majeure partie du temps de reconstruction ?
- Comment intégrer le code dans le cadre ?
- **Comment rendre le code portable ?**
- Comment s'assurer que les threads de l'unité centrale ne sont pas inactifs en attendant le GPU ?
- Comment équilibrer le nombre et le type de GPU avec la fraction de code GPU dans le logiciel ?
- Comment faire correspondre les besoins en GPU du travail avec les machines disponibles sur la grille ?
- **Comment obtenir des résultats reproductibles ?**

Portability

Portability

Supposons que nous ayons réussi à porter le code : 🍷



NVIDIA reste l'acteur principal. Le code hétérogène est généralement écrit en CUDA.

Quelques problèmes:

1. La plupart des sites (CERN, WLCG) **n'utilisent pas de GPU** ... et nous voulons de toutes façons pouvoir fonctionner sur CPU aussi, au cas où.

2. L'adoption des GPU **de nombreux fournisseurs** dans les centres HPC (et pas seulement) est en augmentation. Par exemple :

    - **LUMI-G**, en Finlande, et Frontier, à Oak Ridge, GPU AMD MI250X.
    - **Aurora**, Argonne National Laboratory, GPU Intel Xe.
    - **IDRIS**, Jean Zay H100.

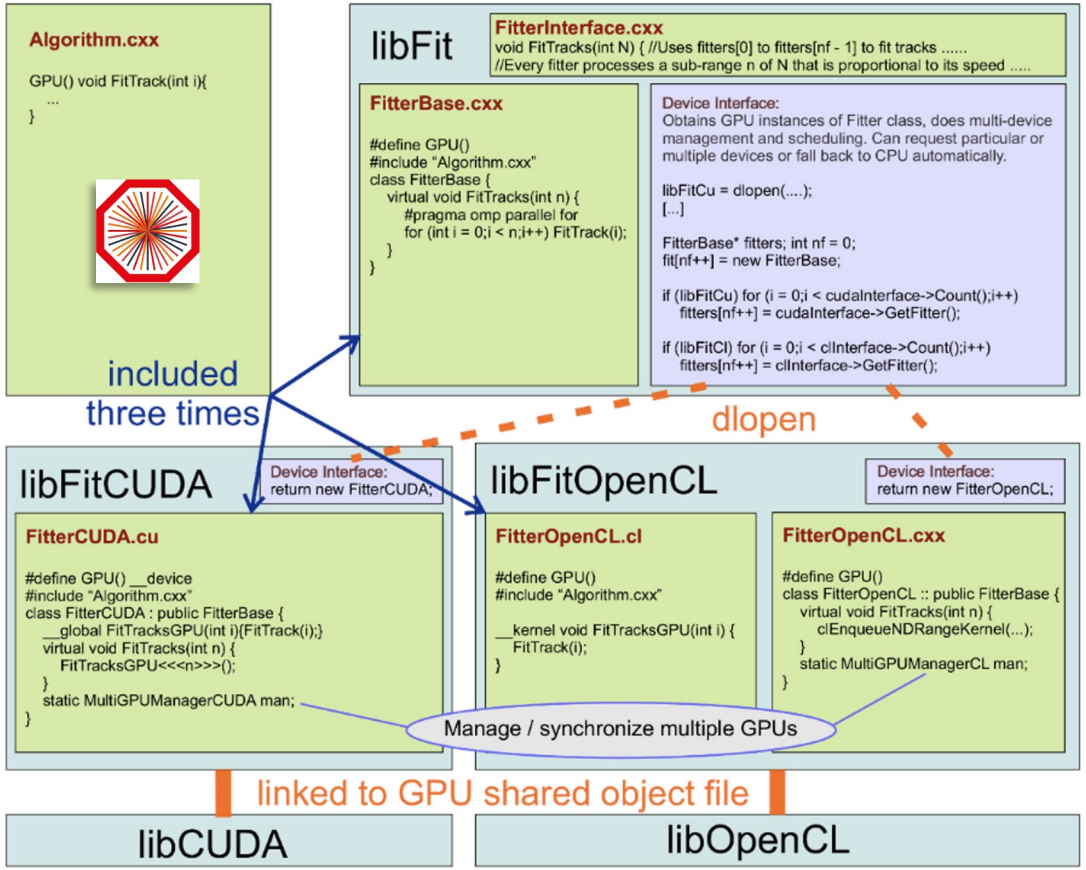
3. **Monopole == mal.**

ALICE et CMS : une approche similaire

Approche “à la main”:

- kernels communs;
- différents wrappers;
- beaucoup des **#ifdefs;**

Ok(-ish) ...



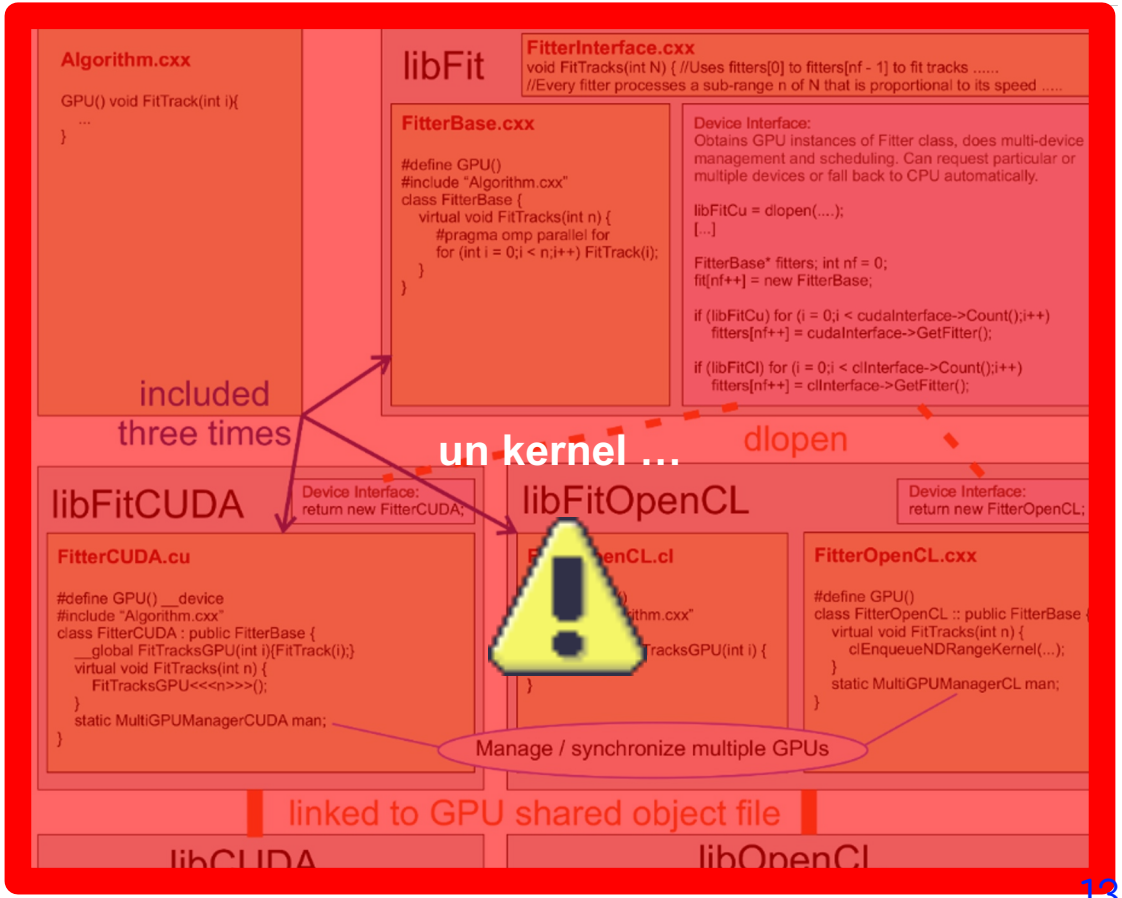
ALICE et CMS : une approche similaire

Approche "à la main":

- kernels communs;
- différents wrappers;
- beaucoup des **#ifdefs;**

Ok(-ish) ... mais:

- **duplication;**
- **gros efforts de validation;**
- **lisibilité tres difficile.**



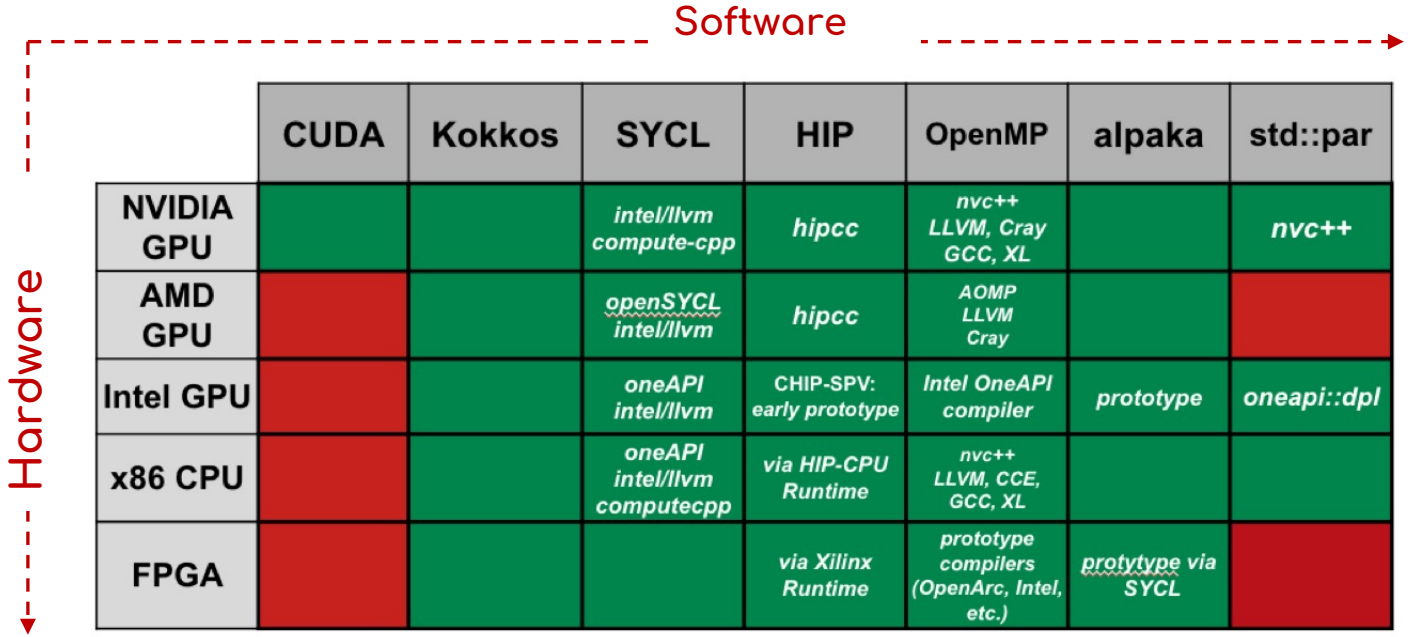
. . . peut-on faire mieux ?

Performance Portability Layers

Performance portability layers:

- **Portability**: plusieurs plates-formes avec un minimum de modifications du code.
- **Performance**: le faire de manière efficace, en retenant les performances natives.

Différents outils avec différentes approches.



	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU			intel/llvm compute-cpp	hipcc	nvc++ LLVM, Cray GCC, XL		nvc++
AMD GPU			openSYCL intel/llvm	hipcc	AOMP LLVM Cray		
Intel GPU			oneAPI intel/llvm	CHIP-SPV: early prototype	Intel OneAPI compiler	prototype	oneapi::dpl
x86 CPU			oneAPI intel/llvm computecpp	via HIP-CPU Runtime	nvc++ LLVM, CCE, GCC, XL		
FPGA				via Xilinx Runtime	prototype compilers (OpenArc, Intel, etc.)	protytype via SYCL	

Performance Portability: Solutions - I

- Language extension: SYCL, std::par.


```
#include <CL/sycl.hpp>

auto p2r_kernels = [=,btracksPtr     = trcks.data(),
                      outtracksPtr   = outtrcks.data(),
                      bhitsPtr       = hits.data()] (sycl::id<1> i) {
                           propagateToR<N>(…)
                           KalmanUpdate<N(…);
                            };
cq.submit([&](sycl::handler &h){
   h.parallel_for(sycl::nd_range(global_range,local_range), p2r_kernels);
});
```

std::par

```
auto p2r_kernels = [=,btracksPtr     = trcks.data(),
                      outtracksPtr   = outtrcks.data(),
                      bhitsPtr       = hits.data()] (const auto i) {
                           propagateToR<N>(…);
                            KalmanUpdate<N(…);
                     };

std::for_each(policy,
              impl::counting_iterator(0),
              impl::counting_iterator(outer_loop_range),
              p2r_kernels);
```

- Compiler pragma-based: OpenMP, OpenACC.

```
#pragma acc parallel loop gang worker collapse(2) \
     default(present) num_workers(NUM_WORKERS)      \
     private(errorProp, temp, rotT00, rotT01
    for (size_t ie=0;ie<nevts;++ie) { // loop over events
       for (size_t ib=0;ib<nb;++ib) { // loop over tracks
          const MPTRK* btracks = bTk(trk, ie, ib);
          MPTRK* obtracks = bTk(outtrk, ie, ib);
          for(size_t layer=0; layer<nlayer; ++layer) {
            const MPHIT* bhits = bHit(hit, ie, ib, layer);
            propagateToR(…);
            KalmanUpdate(…);
          }
        }
      }
  }
```

OpenACC

```
#pragma omp target update to(trk[], hit[])
 nowait depend(out:trk[])

#pragma omp target teams distribute parallel \ for
num_teams(...) num_threads(...) collapse(2)\
  map(to: trk[...], hit[], outtrk[])\
 nowait depend(in:trk[]) depend(out:outtrk[])

for (size_t ib=0;ib<nb;++ib) { // loop over blocks
    for (size_t tIdx=0;tIdx<bsize;++tIdx) { // threads
        #pragma unroll
        for(size_t layer=0; layer<nlayer; ++layer) {
            propagatetoz(...);
            kalmanupdate(...);
        } } }
```

OpenMP

Performance Portability: Solutions - II

- Template Libraries: Alpaka, Kokkos.

```
template <int bSize, int layers, typename member_type>
KOKKOS_FUNCTION void launch_p2r_kernel(
           const member_type& teamMember){
  Kokkos::parallel_for(Kokkos::TeamThreadRange(teamMember,
       teamMember.team_size()),[&] (const int& i_local){
  int i = teamMember.league_rank () * teamMember.team_size ()
            + i_local;

  for(int layer = 0; layer < layers; ++layer) {
      propagateToR<N>(…);
      KalmanUpdate<N>(…);
  });
  return;
}
Kokkos::parallel_for("Kernel",
   team_policy(team_policy_range,team_size,vector_size),
   KOKKOS_LAMBDA( const member_type &teamMember){
       launch_p2r_kernel<bsize, nlayer>());
});
```

okkos

```
struct GPUsequenceKernel
{
public:
    template<typename TAcc>
    ALPAKA_FN_ACC auto operator()(
        TAcc const& acc,
        MPTRK* btracks_,
        MPHIT* bhits_,
        MPTRK* obtracks_
        ) const -> void
    {
       using Dim = alpaka::Dim<TAcc>;
       using Idx = alpaka::Idx<TAcc>;
       using Vec = alpaka::Vec<Dim, Idx>;

   for(int layer = 0; layer < nlayer; ++layer) {
       propagateToR<N>(…);
       KalmanUpdate<N>(…);
     }
};

   alpaka::enqueue(queue, GPUsequenceKernel);
   alpaka::wait(queue);
```

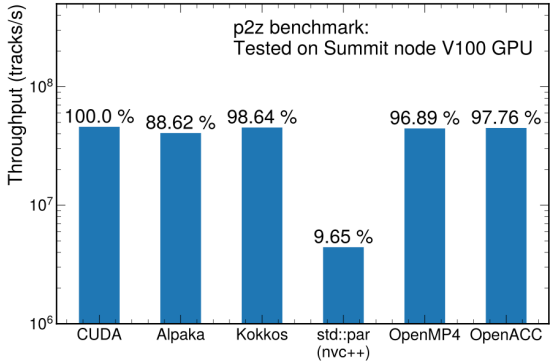
alpaka

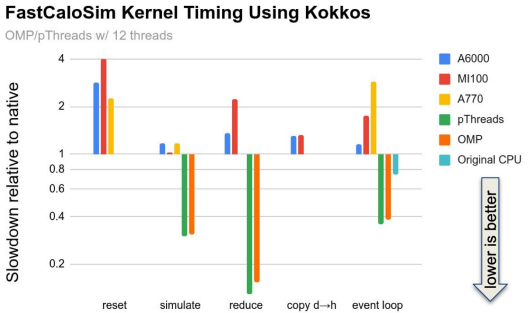
Performance Portability: Benchmarks

Effort commun entre les expériences (HEP-CCE)

	Kokkos	SYCL	OpenMP	Alpaka	std::par
Patatrack	Done	Done*	WIP	Done*	Done compiler bugs
Wirecell	Done	Done	Done	no	Done
FastCaloSim	Done	Done	Done	Done	Done
P2R	done	Done	OpenACC	Done	Done

Évaluation de chaque expérience du portage en fonction d'un certains nombres de paramètres **objectifs** et **subjectifs**.





En général, Kokkos et Alpaka se comportent mieux et sont (plus) simples à intégrer.

- Il n'y a pas encore d'orientation claire pour le moment : trop de facteurs **(courbe d'apprentissage, facilité d'intégration, décisions des fournisseurs)**.

Exemple en production: Alpaka + CMS

Utilisé en 2024 pour la prise de données de CMS.

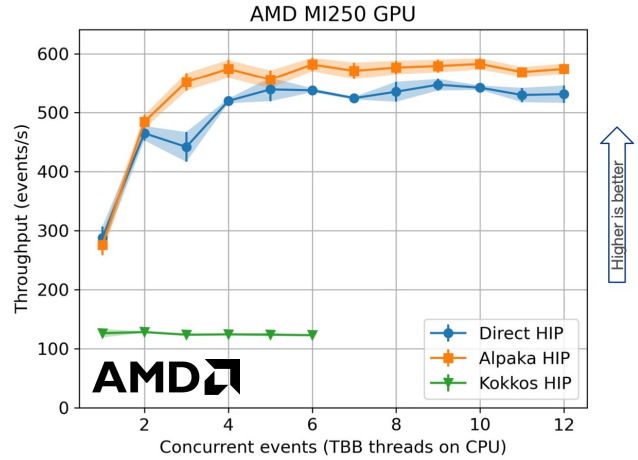
Bonnes performances sur l'hardware actuel (T4 + AMD Milan).

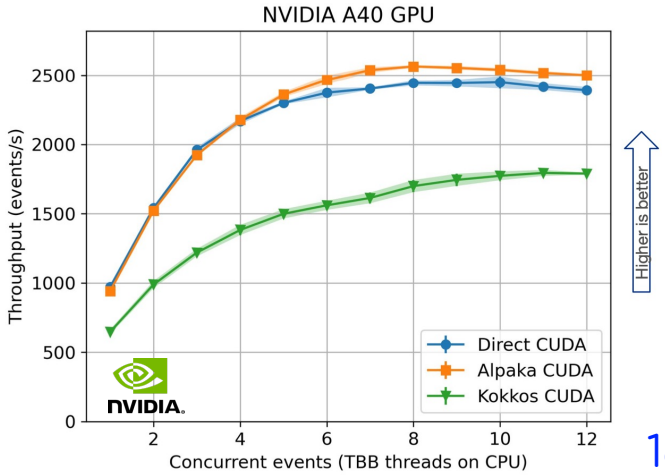
Librairie header-only, facile à intégrer:

- supporte le multithreading (host);
- supporte plusieurs backends avec une compilation:
  - GPU de différents fournisseurs et de différentes générations;
  - CPU avec différents modes d'exécution, ex. l'exécution parallèle en utilisant TBB.

Approche low-level, très proche de CUDA.

- **Facile: portage CUDA + enseigner aux nouveaux arrivants.**





Reproducibility

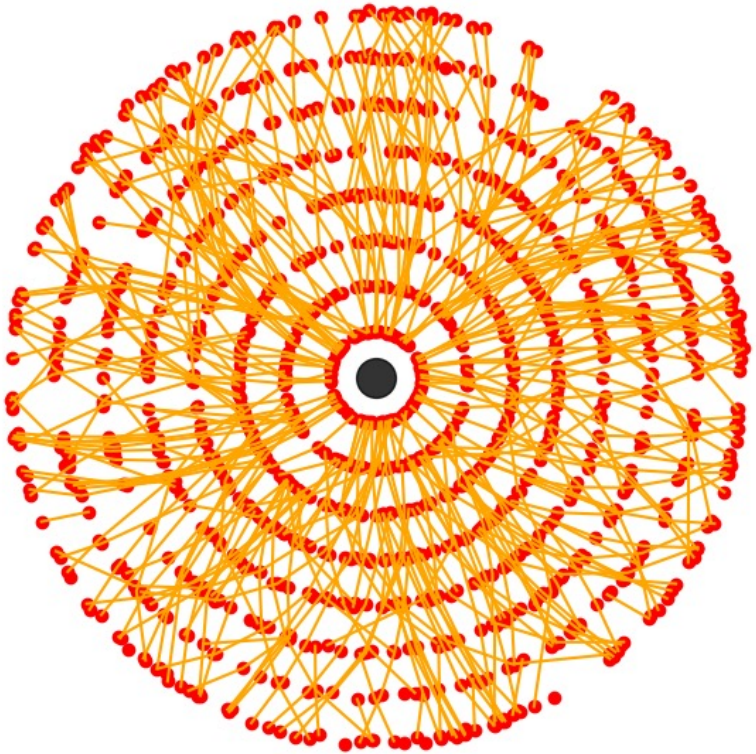
Caveat Emptor

Les algorithmes parallèles posent des problèmes supplémentaires par rapport aux algorithmes séquentiels:

- plus complexes à concevoir et à mettre en œuvre efficacement;

- **l'ordre d'exécution non défini peut produire des résultats qui ne sont pas entièrement reproductibles.**

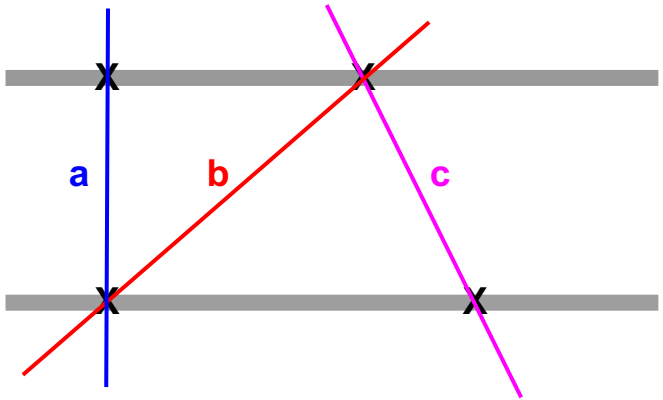
Différences intrinsèques : ex. reconstruction de traces



Différences intrinsèques : ex. reconstruction de traces

Les **traces** partageant un ou plusieurs « points » sont considérées comme **ambiguës** : une seule est bonne et les autres sont fausses.

Les tâches GPU sont exécutées **dans un ordre aléatoire**. La résolution de l'ambiguïté dépend de l'ordre et le résultat n'est donc pas déterministe.



Supposons que nous ayons trois traces de « qualitè » **q(c)>q(b)>q(a)**. Nous pouvons obtenir des résultats différents en fonction de l'ordre :

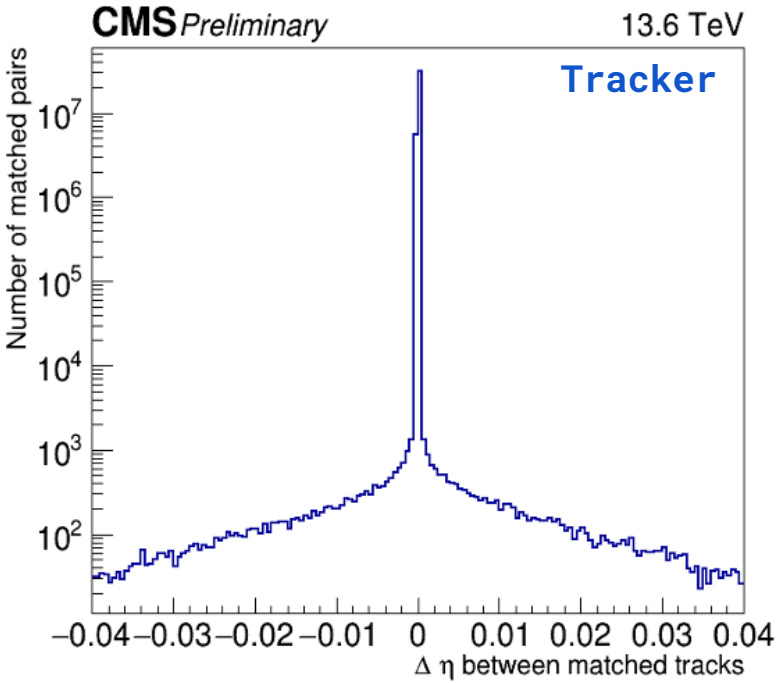
1) **b** vs **c** → **b** KO ⇝ **a** et **c** OK!

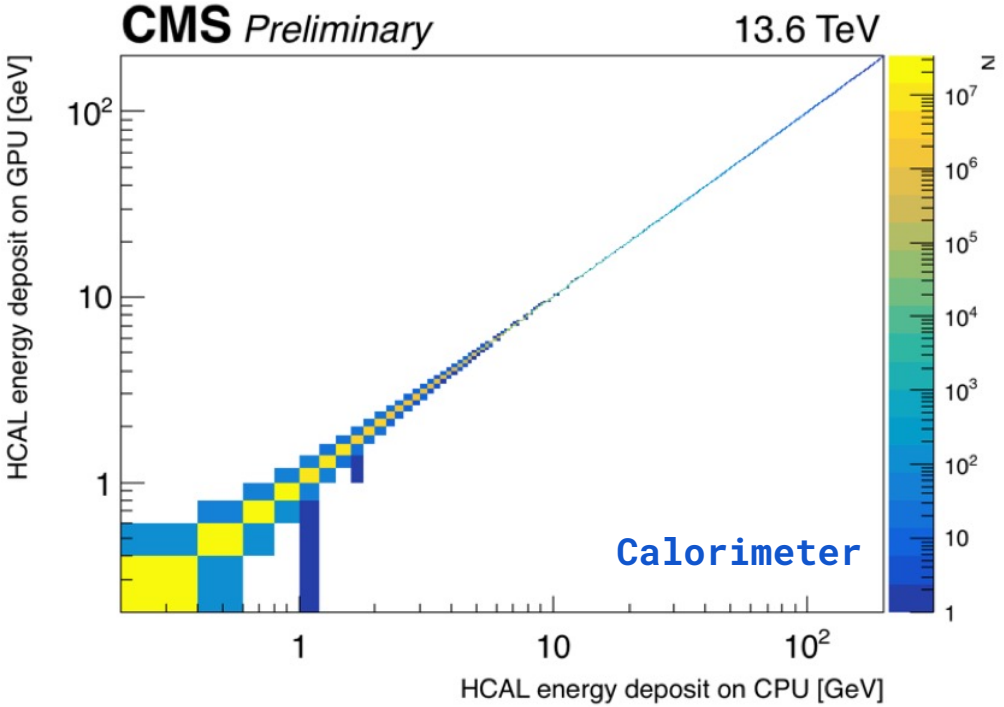
2) **a** vs **b** → **a** KO; **c** vs **b** → **b** KO ⇝ **c** OK!

N.B. On pourrait rendre le système déterministe d'une manière ou d'une autre. Mais cela le rendrait très probablement beaucoup plus lent. Ce n'est pas impossible. C'est un compromis. Il n'y a pas de « free lunch ».

Quelques chiffres réels: comparaisons événement par événement

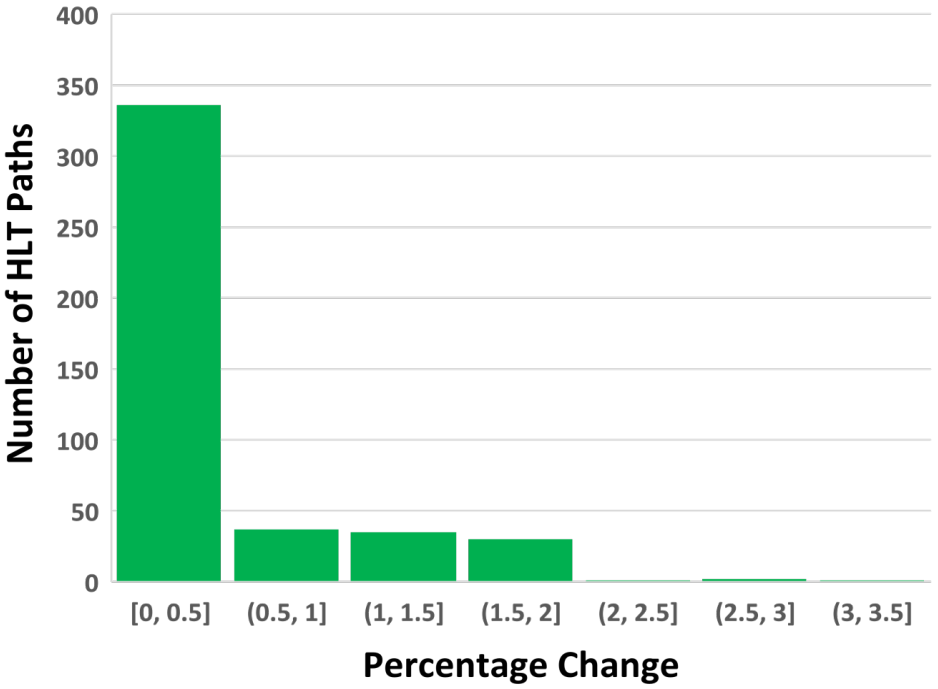
- Data Quality Monitoring (DQM) pour comparer les différences en ligne.
- Gros effort pour réduire les différences CPU / GPU.
- Les différences actuelles entre les objets CPU/GPU sont très faibles . . .





Quelques chiffres réels: les effets sur la sélection des événements

Différences, événement par événement, - obtenu après avoir exécuté le trigger avec et sans GPU (le trigger de CMS est en pratique un ensemble de filtres):



1.28 M événements

- En considérant les filtres qui sélectionnent **plus de 100 évènements (sur 1,28 M)** : 99% des filtres ont une différence inférieure à 2 %.

Un filtre spécial a été introduit, qui accepte les événements lorsqu'il y a des différences (5316/2,312,690)

- $\epsilon = {}^{5316}/_{2312690} = 0.22\%$

Les différences résiduelles sont en cours de vérification.

Mais tout de même : vers des « incertitudes de mesures » architecturales ?

Conclusions

Le portage du code est (de loin) la partie la plus difficile, mais ce n'est pas la fin de l’histoire.

fin

Des questions?