ICHEP: 28 July - 6 August, 2020

GPU-based online-offline reconstruction in ALICE for LHC Run 3

Matteo Concas, INFN Torino
on behalf of the **ALICE** collaboration

mconcas@cern.ch

INFN
Istituto Nazionale di Fisica Nucleare

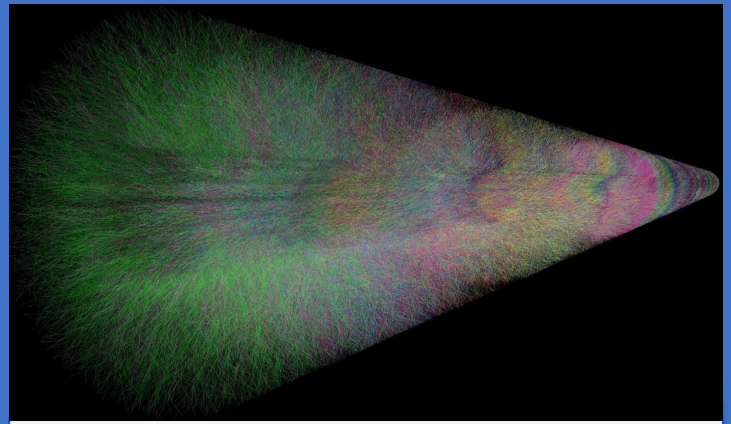
ALICE

ALICE data-taking in Run 3

- Run 3: LHC will deliver **50 kHz in Pb-Pb** collisions
  - ALICE plans to record an integrated luminosity of >10 nb^{-1} with minimum bias
- **Developments and upgrades** of the entire experimental setup **to address the challenge**

ALICE data-taking in Run 3

- Run 3: LHC will deliver **50 kHz in Pb-Pb** collisions
  - ALICE plans to record an integrated luminosity of >10 nb^{-1} with minimum bias
- Developments and upgrades of the entire experimental setup to address the challenge
- **New triggerless data acquisition: continuous readout**
  - Stream of input data, splitting along time dimension: timeframes (TF) ~ms
  - Overlap of multiple events data in drift detectors
  - 50x more events compared to Run 2 → **50x more data!**

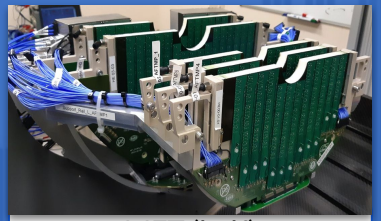


Overlapping events in TPC, @50 kHz Pb-Pb (2 ms)

ALICE data-taking in Run 3

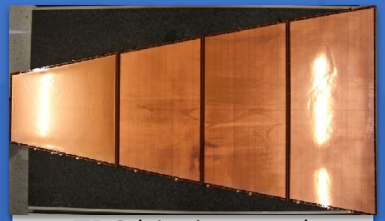
- Run 3: LHC will deliver **50 kHz in Pb-Pb** collisions
  - ALICE plans to record an integrated luminosity to >10 nb^{-1} with minimum bias
- Developments and upgrades of the entire experimental setup to address the challenge
- New triggerless data acquisition approach: continuous readout
- **Upgraded and new detectors**[1]
  - **TPC**: upgraded readout capabilities using **GEM** to cope with 50 kHz in Run 3
  - **ITS**: replaced with new one made of **7 silicon pixel layers**
  - **MFT**: new detector for **forward tracking** in the central barrel
  - **FIT**: upgraded layout and **improved performance**[2]

[1]S.M. Panebianco: "ALICE upgrades for Run 3"



MFT (half)

ITS (half inner barrel)



TPC (single sector)

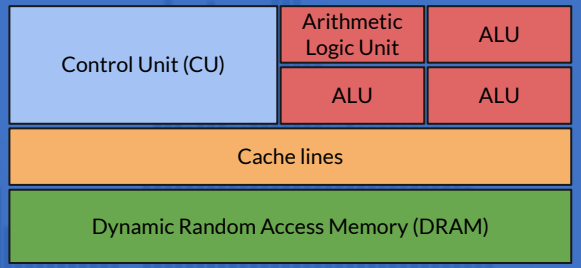
[2]M. Slupecki: "Status of the Fast Interaction Trigger for ALICE Upgrade"

ALICE data-taking in Run 3

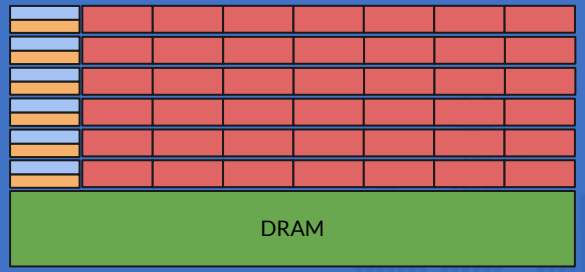
- Run 3: LHC will deliver **50 kHz in Pb-Pb** collisions
  - ALICE plans to record an integrated luminosity to >10 nb^{-1} with minimum bias
- Developments and upgrades of the entire experimental setup to address the challenge
- New triggerless data acquisition approach: continuous readout
- Upgraded and new detectors
- **Need for data compression and no high-level trigger: Online reconstruction**
  - Requires online calibration
  - Extremely demanding task: **large amount of computing power is required**
- **Offload** large part of the reconstruction **on GPU**s to gain speed:
  - Programmable **parallel device** that manages **thousands of threads** compared to hundreds for CPUs
  - Many algorithms efficiently fit the architecture: **high processing throughput → faster global execution**
  - **Convenient**: computing nodes can be equipped with up to 64 CPU cores and up to 8 GPUs each

ALICE data-taking in Run 3

- Run 3: LHC will deliver **50 kHz in Pb-Pb** collisions
  - ALICE plans to record an integrated luminosity to >10 nb^{-1} with minimum bias
- Developments and upgrades of the entire experimental setup to address the challenge
- New triggerless data acquisition approach: continuous readout
- Upgraded and new detectors
- **Need for data compression and no high-level trigger: Online reconstruction**
- **Offload** large part of the reconstruction **on GPU**s to gain speed



Essential layout of CPU components



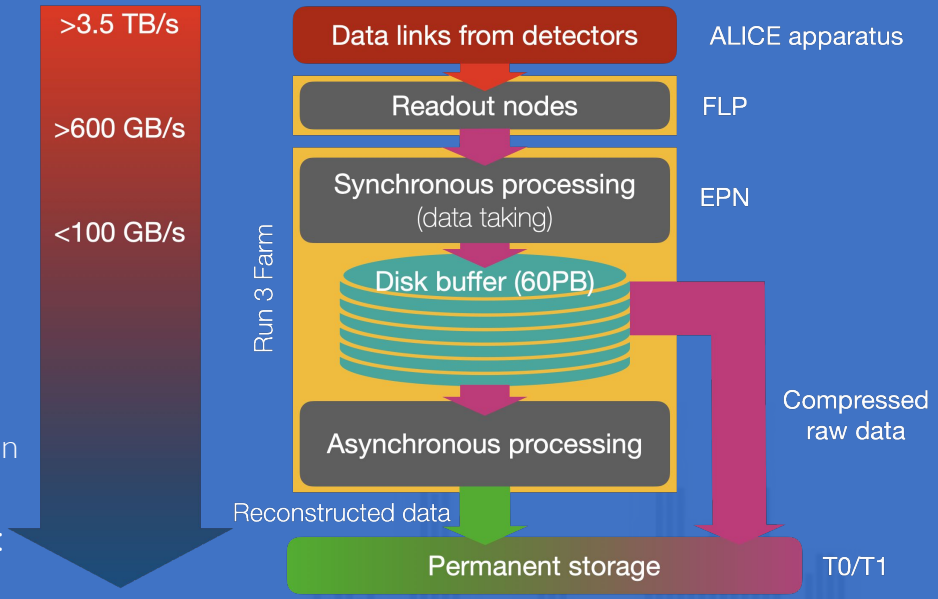
Essential layout of GPU components

Online reconstruction and the EPN farm

- **Online reconstruction on the EPN farm**[1]
  - Events distributed on event processing nodes equipped with GPUs (order of few thousands)
  - Alice **O²** (Online-Offline) framework to steer the process, support for onboard GPUs

- **Synchronous**: during data taking[2]:
  - Main user: TPC, multiple use cases
  - 100% TPC standalone tracking runs on GPUs
  - ITS and TRD tracking on fewer events → calibration

- **Asynchronous**: during no-beam and pp collisions:
  - Full reconstruction including all detectors
  - GPUs used also by ITS and TRD reconstruction
  - Remaining processing can be split on 3x⅓ EPN/T0/T1



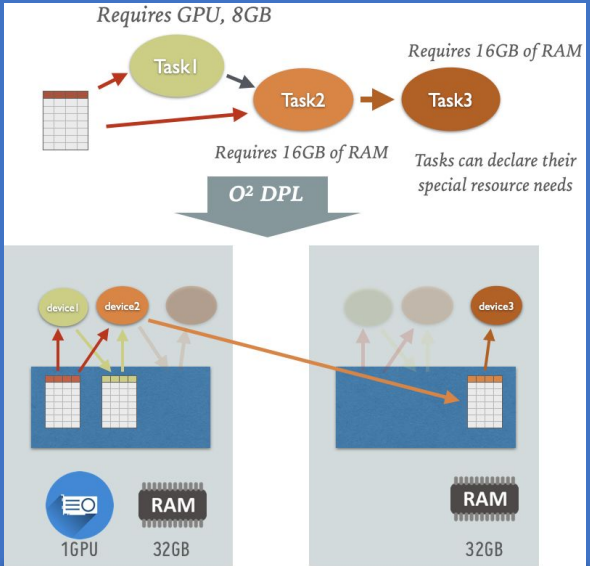
[1]C. Zampolli: "ALICE data processing for Run 3 and Run 4 at the LHC"

[2]M. Lettrich: "Fast Entropy Coding for ALICE Run 3"

GPU integration in O^2

- **O^2**: Single software framework for online and offline computing
  - Implicit workflow description translated by data processing layer (DPL) to a topology of processes
  - Communication across tasks is done via message passing: works locally and distributed across computing nodes

- A DPL device is able to steer GPU executions
  - Multiple processes can share the same GPU and run in parallel
  - Multiple tasks running on different TFs simultaneously: mask the asynchronous behaviour of GPUs

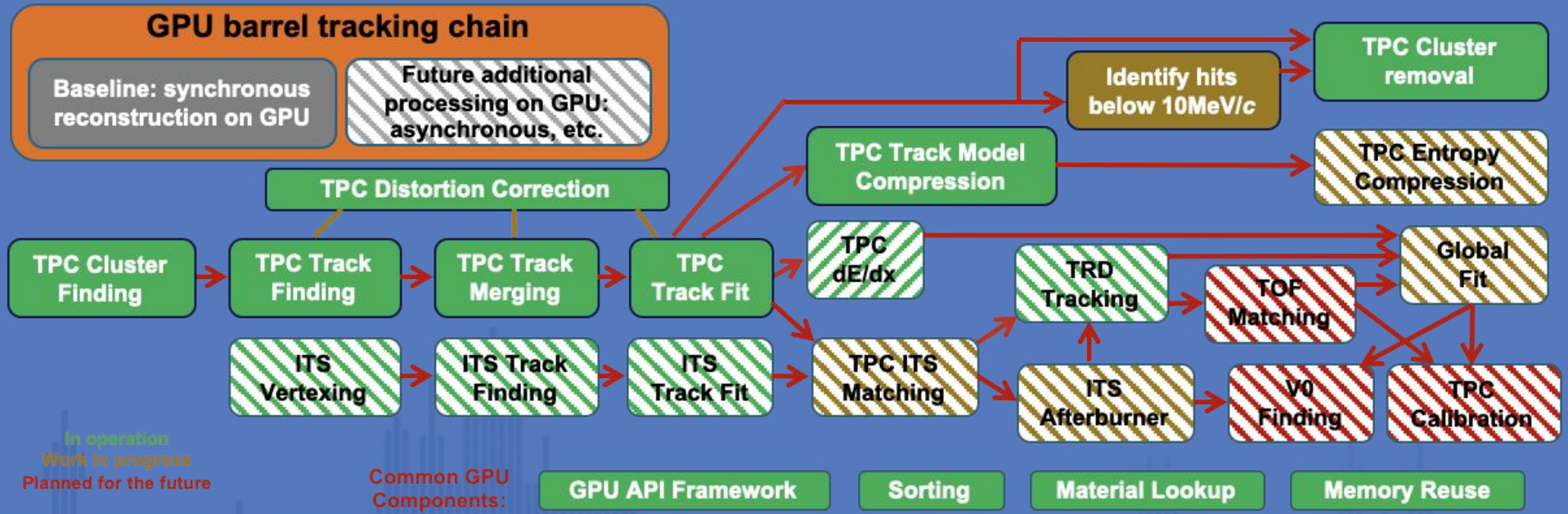


Distributed heterogeneous workflow using DPL

GPU: current and future scenarios

- Now: ALICE is ready to run a **baseline** scenario
  - Keep up with the data rate online
  - TPC tracking on GPU: resources are fully exploited
- More GPUs available in asynchronous phase: aim for an **optimistic** scenario:
  - As much as possible for the beginning of Run 3
  - Offload as much as possible the computation on GPU, more heterogeneous resources available
  - Other detectors have online reconstruction process implemented on GPUs
- **Ideally: the whole barrel tracking carried out on GPU**
  - Some intermediate state will work already

ALICE GPU processing: state of the art



- Baseline scenario is fully operative
- Common GPU API framework to steer all the reconstruction on GPU
- Multiple programming interfaces to different platforms are supported

Result consistency and code portability

- **Consistency of the results** obtained with different platforms (CPU/GPUs)
  - Numerical deviations due to different optimisations and order of execution of the instructions in mathematical functions
  - Same algorithms across different implementations: parallel approach affecting results with negligible effects

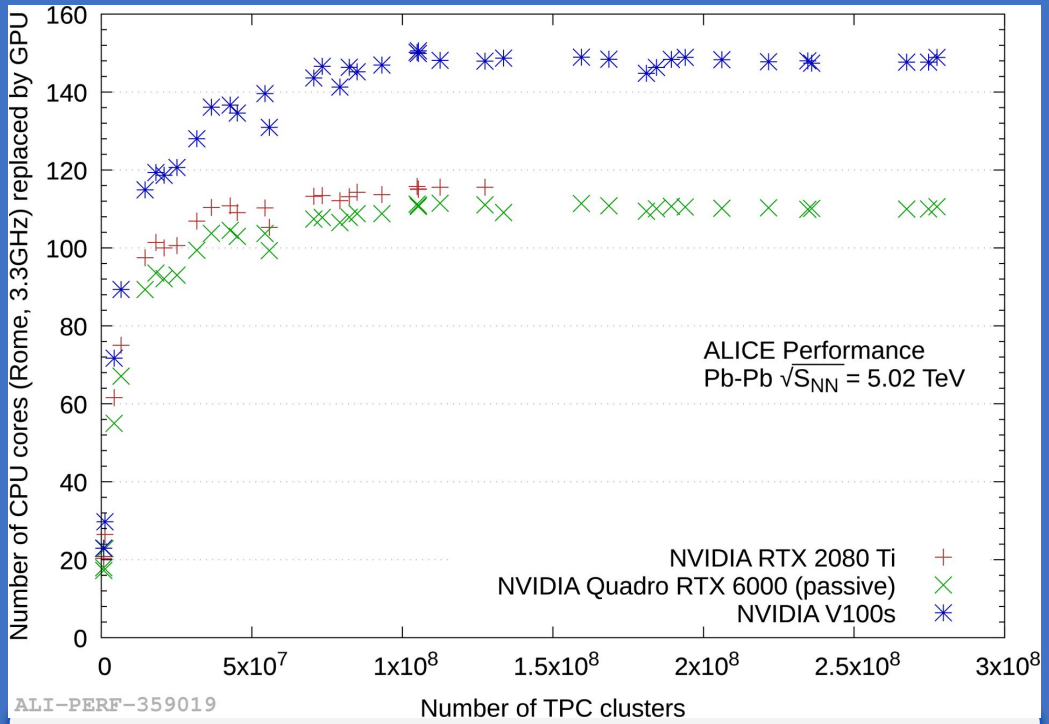
Result consistency and code portability

- **Consistency of the results** obtained with different platforms (CPU/GPUs)
  - Numerical deviations due to different optimisations and order of execution of the instructions in mathematical functions
  - Same algorithms across different implementations: parallel approach not affecting results

- **Portability of the code** achieved by a single transparent interface
  - **Support multiple platforms** such as CPU and different GPU brands (Nvidia and AMD at the moment)
  - Maintaining a **single code is better** than replicating it for different devices
  - Adapt to the correct underlying architecture available on the target host via **dynamic loading of the required libraries**

GPU tracking: results and performance

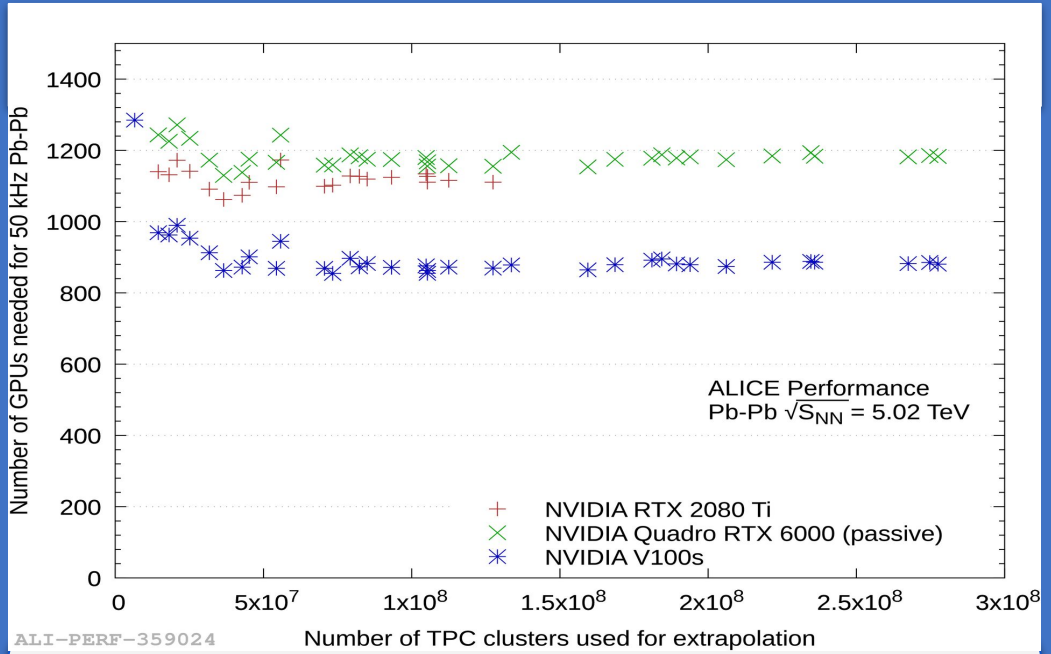
- **Efficient usage of these accelerators allows us to trade from 40 to 150 CPU cores with a single graphics card**
  - The speedup depends on the algorithm. This average contains some parts not optimized for CPU, where speedup is 200-300.
  - For TPC tracking and fit speedup is ~50-100



Average speedup normalised to one core for different GPU cards and a comparison with former offline tracking vs HLT tracking.

GPU tracking: results and performance

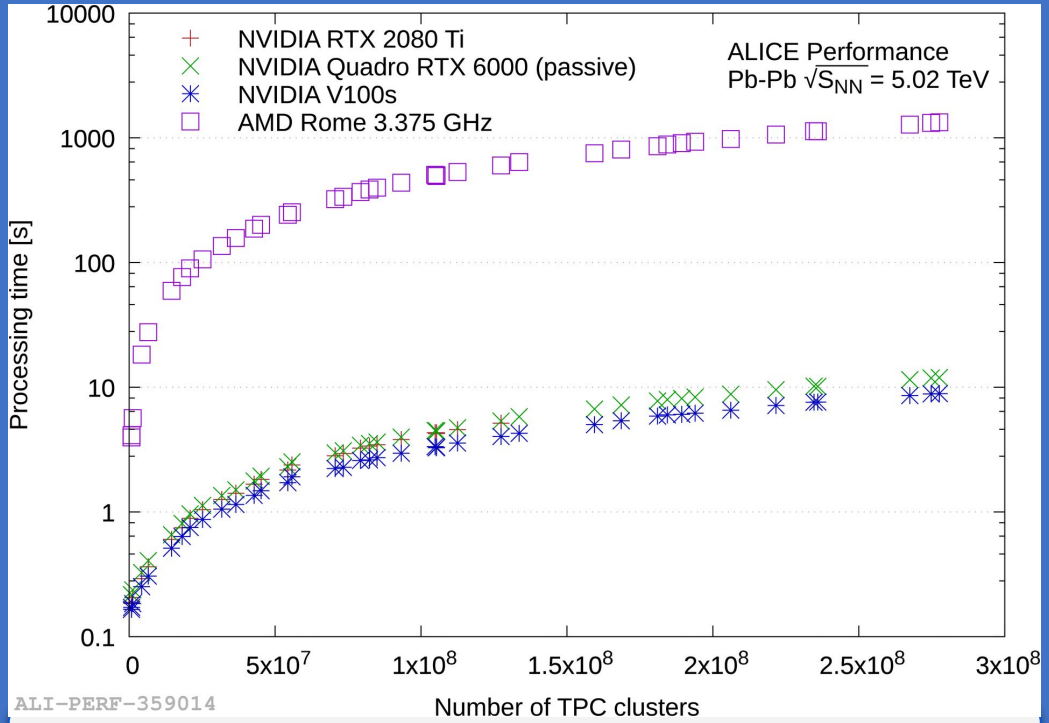
- Efficient usage of these accelerators allows us to trade from 40 to 150 CPU cores with a single graphics card
- **Number of required GPUs is ~constant**



Processing time as a function of the input data size, obtained by varying the length of a time frame in continuous readout regime

GPU tracking: results and performance

- Efficient usage of these accelerators allows us to trade from 40 to 150 CPU cores with a single graphics card
- Number of required GPUs is ~constant
- **The processing time scales linearly with the data size: in continuous readout regime, adjustments on the timeframe length will not affect performance**
  - In the most relevant case, the TPC reconstruction chain, the speedup reaches a factor ~100



Comparisons for the computing time spent between recent CPU and GPUs, two orders of magnitude faster

Conclusions and outlook

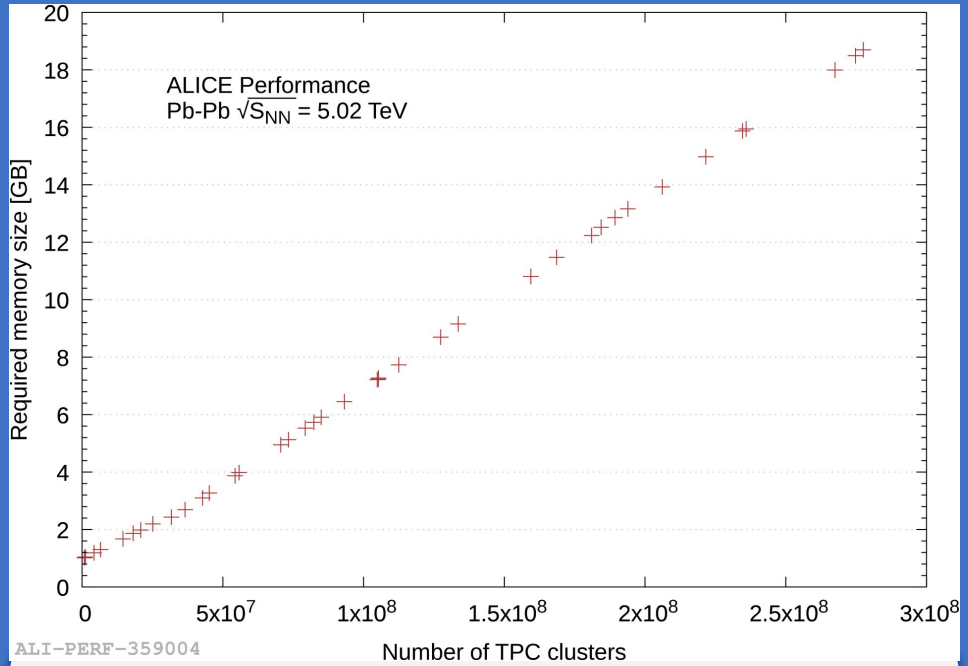
- In Run 3 ALICE will take **50 kHz of Pb-Pb** data in a triggerless regime: **continuous readout**
- High **data compression** factors require **online reconstruction**
  - **GPUs** are the **pivotal architecture** to afford the data reduction
- **TPC and ITS reconstruction on GPU is the basic goal,** ideally: whole barrel tracking on GPU
- **TPC: most relevant use case**, its speed-up on GPU is a factor of x10. One GPU replaces 40-150 CPU cores

- Foreseen to enqueue **more** reconstruction on **GPUs during asynchronous phase**
- Investigating **broader usage** of GPUs **in the future**
  - Expertise and knowledge base are growing → plans to expand the range of applications
  - Not only tracking or **reconstruction** → **digitization** (simulations) and **analysis**
- Keep an eye on the **future of HEP and HPC also in view of next runs at LHC**
  - Grid sites could in future be equipped with GPUs

Backup

Memory performance

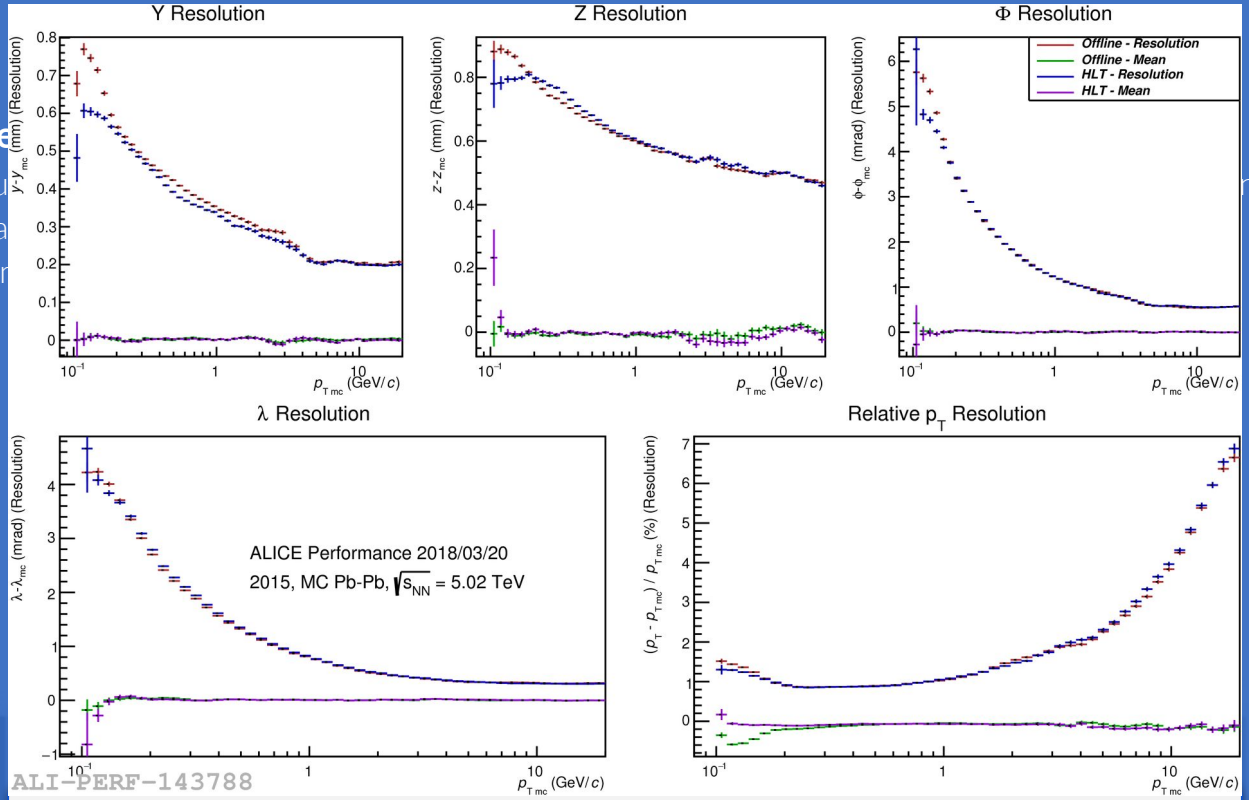
- Memory required for reconstruction scales linearly with the number of TPC clusters: most demanding use case drives the benchmarks trendings



The required memory scales linearly with the size of the input data

Code portability and result consistency

- **Consiste**
  - Nu… n
    ma…
  - Sar…



Result consistency between CPU (Offline, in legend) and GPU (HLT, in legend)

Result consistency and code portability

- **Consistency of t**
  - Numerical d … uctions in mathematic
  - Same algorit … ult
- **Portability of th**
  - Support mul … t the moment)
  - Maintain a s
  - Adapt to the … oading

Algorithm.cxx

```
GPU() void FitTrack(int i){
   ...
}
```

included three times

libFit

FitterInterface.cxx

```
void FitTracks(int N) { //Uses fitters[0] to fitters[nf - 1] to fit tracks .....
//Every fitter processes a sub-range n of N that is proportional to its speed .....
```

FitterBase.cxx

```
#define GPU()
#include "Algorithm.cxx"
class FitterBase {
   virtual void FitTracks(int n) {
      #pragma omp parallel for
      for (int i = 0;i < n;i++) FitTrack(i);
   }
}
```

Device Interface:
Obtains GPU instances of Fitter class, does multi-device management and scheduling. Can request particular or multiple devices or fall back to CPU automatically.

```
libFitCu = dlopen(....);
[...]

FitterBase* fitters; int nf = 0;
fit[nf++] = new FitterBase;

if (libFitCu) for (i = 0;i < cudaInterface->Count();i++)
   fitters[nf++] = cudaInterface->GetFitter();

if (libFitCl) for (i = 0;i < clInterface->Count();i++)
   fitters[nf++] = clInterface->GetFitter();
```

dlopen

libFitCUDA

Device Interface:
return new FitterCUDA;

FitterCUDA.cu

```
#define GPU() __device
#include "Algorithm.cxx"
class FitterCUDA : public FitterBase {
   __global FitTracksGPU(int i){FitTrack(i);}
   virtual void FitTracks(int n) {
      FitTracksGPU<<<n>>>();
   }
   static MultiGPUManagerCUDA man;
}
```

libFitOpenCL

Device Interface:
return new FitterOpenCL;

FitterOpenCL.cl

```
#define GPU()
#include "Algorithm.cxx"

__kernel void FitTracksGPU(int i) {
   FitTrack(i);
}
```

FitterOpenCL.cxx

```
#define GPU()
class FitterOpenCL :: public FitterBase {
   virtual void FitTracks(int n) {
      clEnqueueNDRangeKernel(...);
   }
   static MultiGPUManagerCL man;
}
```

Manage / synchronize multiple GPUs

linked to GPU shared object file

libCUDA | **libOpenCL**

Produced libraries for different platforms, enabled via dynamic load by generic interface