MATTEO CONCAS - POLITECNICO DI TORINO (DET)
"WORKSHOP DI CCR" JUNE 3-7, 2019

GPUS IN RUN 3: A SHIFT IN THE OFFLINE/ ONLINE PROCESSING PARADIGM FOR ALICE

OVERVIEW

- The goal of this presentation is to show current state of the art of the available technologies used and/or investigated in ALICE during the development reconstruction software for the upgrade. Some things are stable already, others are likely to change, things move fast
- This presentation includes and mentions the work of many in the collaboration. Credits and reference are added whenever possible. Any consideration or opinion that might transpire during this presentation has to be considered as my personal

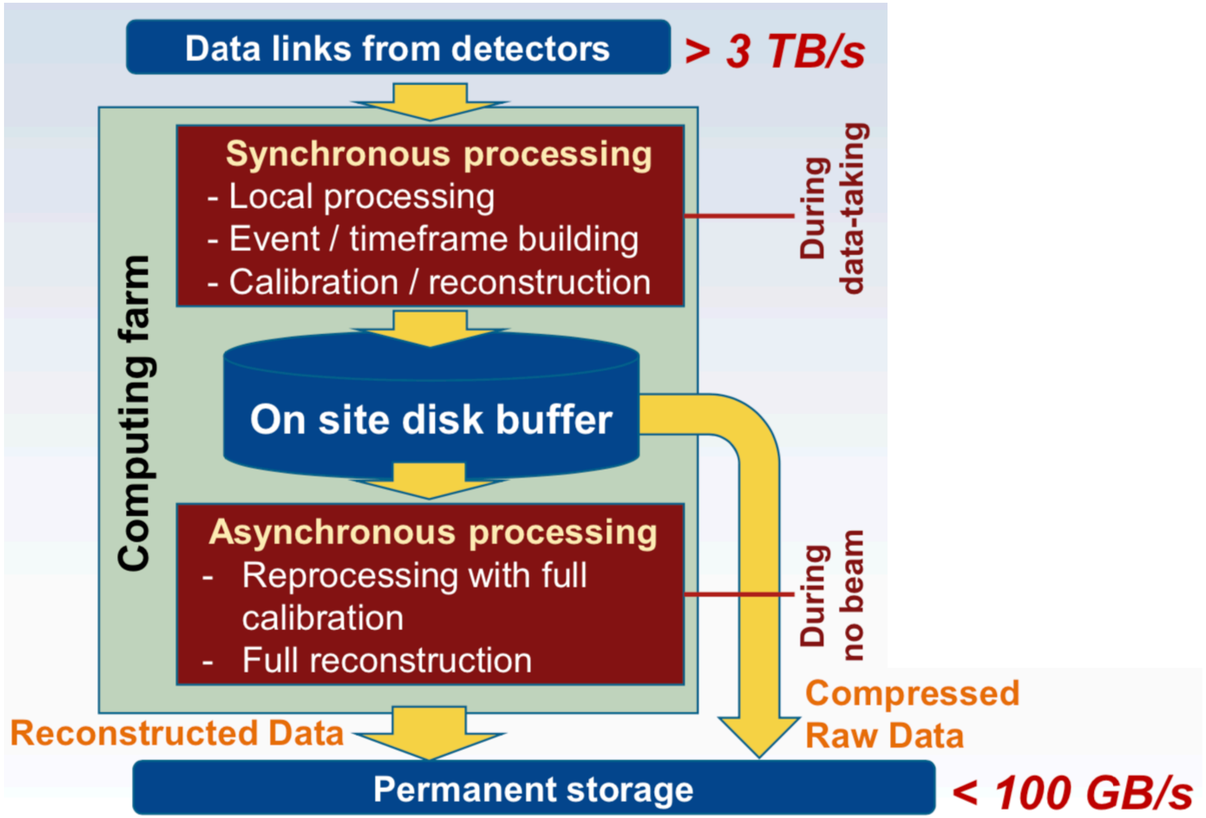
RUN 3 AT THE LARGE HADRON COLLIDER (LHC)

- The accelerator will deliver an increased luminosity → $6x10^{27}cm^{-2}s^{-1}$, up to 50kHz in PbPb collisions
- Some experiments will dramatically improve their capabilities, both on the hardware and software
- A Large Ion Collider Experiment (ALICE), among them:
  - will replace its innermost detector, the Inner Tracking System (ITS) with a brand new one, entirely based on silicon pixel detectors, with better pointing resolution*, tracking efficiency at low p_T[1]
  - will introduce a continuous readout mode, possibly partitioning data in so-called *timeframes*
  - will introduce: O^2 a renewed software stack for simulation, reconstruction and analysis written from scratch, with a multi-process structure ready to scale across clusters. The structure is granularly divided in *devices,* which represent individual compute tasks, that communicates via the abstraction of transport, transparently supporting different communication strategies[8] (shared memory, ZeroMQ)

*increased by a factor of 3(5) in rφ (z) at p_T=500 MeV/c

THE ONLINE/OFFLINE RECONSTRUCTION IN ALICE: CHALLENGES AND GOALS

- Run2 vs Run3 comparison
  - move from O(1) kHz single events (triggered) up to 50kHz of continuous data acquisition
  - reconstruct up to 50x more events
  - not enough storage: need for data reduction and data compression, from the nominal 3+ TB/s to less than 100 GB/s
  - from the division from online "*quick & dirty*" and offline "*Precise but slow*" reconstruction paradigm towards a "synchronous vs asynchronous" reconstruction sharing the same software codebase



D. Rohr - "Track Reconstruction in the ALICE TPC using GPUs for LHC Run 3" Connect the Dots 2018"[2]

HOW TO ACHIEVE THE NEEDED DATA REDUCTION: ONLINE RECONSTRUCTION

- With continuous readout we need to be able to perform calibration and full reconstruction to quickly select physically interesting "*events*" to be stored. The reconstruction is divided in two phases
  - Synchronous (during data taking): perform online calibration and data compression
  - Asynchronous (during no beam time): full reconstruction with final calibration
- The most computational-demanding phases are proportional to powers of the event multiplicity e.g. in tracking and vertex reconstruction, because of the presence of heavy combinatorial sections in the algorithms (~6000 charged particles produced in the acceptance)
- In most of cases the *event* processing can be trivially split across parallel unrelated computations, the problem is often embarrassingly parallel → scales with the number of used computing units the approach is limited by the amount of RAM required by the process
- Parallel accelerators such as GPUs allows us to exploit the high core/threads density and the dedicated memory to address the computing demand, both releasing memory on the host that can be used by other tasks and adding resources to the same host
- Graphic Processing Units (GPUs) are a suitable choice (General Purpose GPU Computing) → Introduction in the O^2 the possibility to run the reconstruction on GPUs alongside the generic code that can runs on CPU (OpenMP, C++11 threads, OpenCL, …)

GPU IN RECONSTRUCTION IN ALICE: USED TECHNOLOGIES AND OUTLOOK

- GPU market split between two major actors AMD and Nvidia*, nowadays is not yet known whether ALICE will adopt GPUs and which architecture would be chosen in case. It would be anyway optimal to be ready also to exploit different GPU architectures also considering the reconstruction of Montecarlo generated events which involve the same procedure
- GPGPU programming languages taken in consideration
  - CUDA[3]: proprietary, closed-source API to program against Nvidia GPUs.
    Pros: At the moment leader in the innovation front, capable to exploit each new feature in latest architectures, by construction
    Drawback: vendor lock-in
  - OpenCL[4]: the open, royalty-free standard for cross-platform language, by Khronos group. At the moment we are using the v1.2 for both TPC and have ITS standalone version of the tracking code
    Pros: can run on different architectures: CPU, GPU, FPGA… → single codebase, in principle easier to maintain
    Drawbacks: generally support a subset of the features available for CUDA, this is mainly because of the portability of the interface which needs to be compatible with diverse architectures and may not allow to be tailored to fit only one of them. Real performances may vary from one architecture to another
  - HIP[5] (Heterogeneous-Compute Interface for Portability, with ROCm): [*evaluation in progress*] C++ runtime API and kernel language that allows developers to create portable applications that can run on AMD and other GPUs.
    Pros: got some boost in development, stimulated by the need to stay in the market; possible to semantically convert CUDA kernels code to HIP API calls making it possible to map CUDA kernels on software ready to be deployed on AMD GPUs; can dramatically reduce the code to be maintained, still supporting diverse architectures
    Drawback: always one step behind latest released CUDA features (must be said that eventually the reconstruction code will reach some stable release; if HIP fits the needs once, one does not really need to have regular updates on the translation)

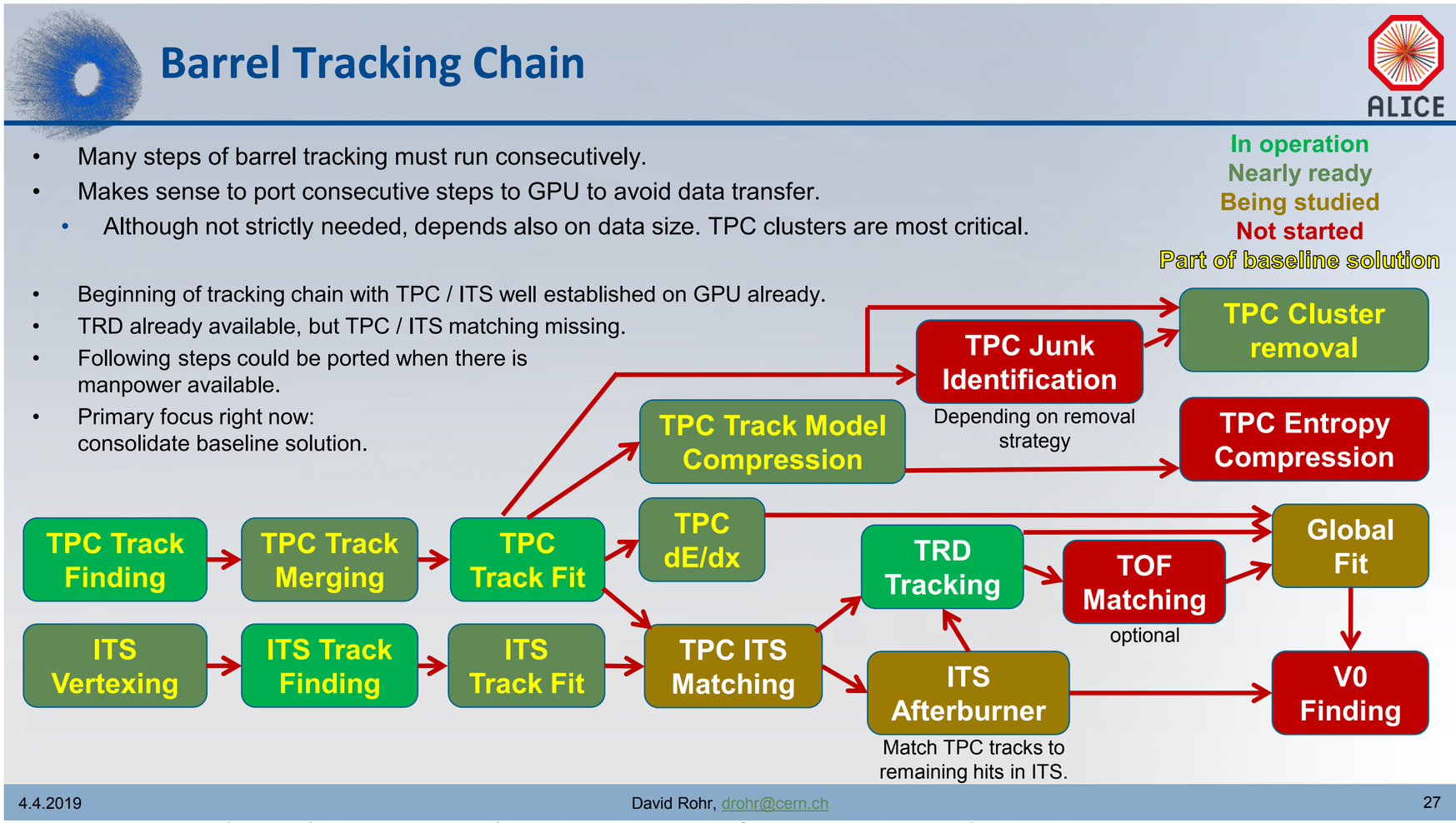
*Intel GPUs might not come in time to join the discussion for Run3

GPU IN RECONSTRUCTION IN ALICE: STATE OF THE ART

- TPC: tracking in the High Level Trigger (HLT) has been already in place since Run2
  - first to port the O^2 version for Run 3, based on Cellular Automaton (CA) and Kalman filter (KF)
  - implementation with OpenMP, CUDA, OpenCL[2], HIP on its way
- ITS: tracking[1] and vertex reconstruction
  - tracking based on CA and KF, vertex reconstruction based on cluster identification, to cope with the pile-up of many events on same bunch-crossing (~5 piled up events in pp collisions with readout base option)
  - parallel implementation using CUDA, OpenCL (standalone tracking version)
- Transition Radiation Detector (TRD) is also using a GPU-accelerated tracking and fitting

GPU TRACKING PERSPECTIVES

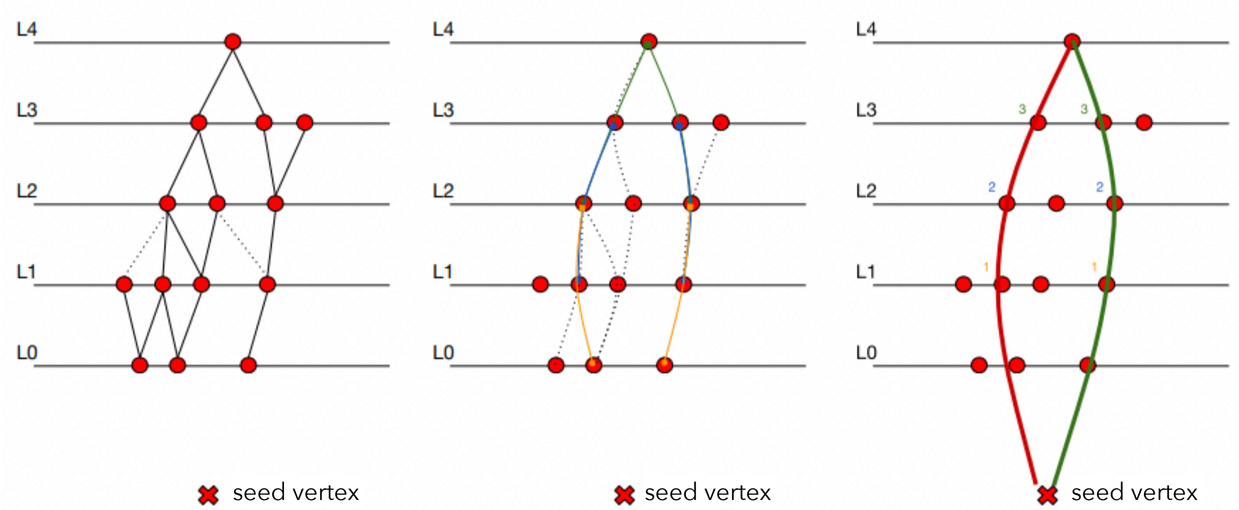
- In a first iteration the different steps in the tracking have been developed separately, to naturally share the workload across experts of different detectors, leaving to them the evaluation and decision of tracking strategies
- With GPU-based workflow, this leads to some obvious overhead, especially in data moving across host and device before and after each phase
- It appears natural to connect steps that share common data structures into *pipelines* on the GPU, to save all those transactions and to avoid useless and expensive transfers/allocations
- On the other hand, the execution of the unrelated and distributable steps are managed by the multi-process nature in O^2



D. Rohr - "Track Reconstruction in the ALICE TPC using GPUs for LHC Run 3" Connect the Dots 2019"[6]

USE-CASE EXAMPLE: TRACK RECONSTRUCTION IN ITS USING CELLULAR AUTOMATA

- The reconstruction in ITS is responsible to find and classify the tracks generated by charged particles and the position of the interaction vertex
- After a preliminary vertex position estimation, needed as a seed for current tracking algorithm implementation, the tracking phase is constituted by three steps



Tracklet finding

A combinatorial routine to find pairs of clusters on adjacent layers, filtering them using some criteria

Cell finding

Subsequent tracklets that satisfy some filtering criteria are merged into cell

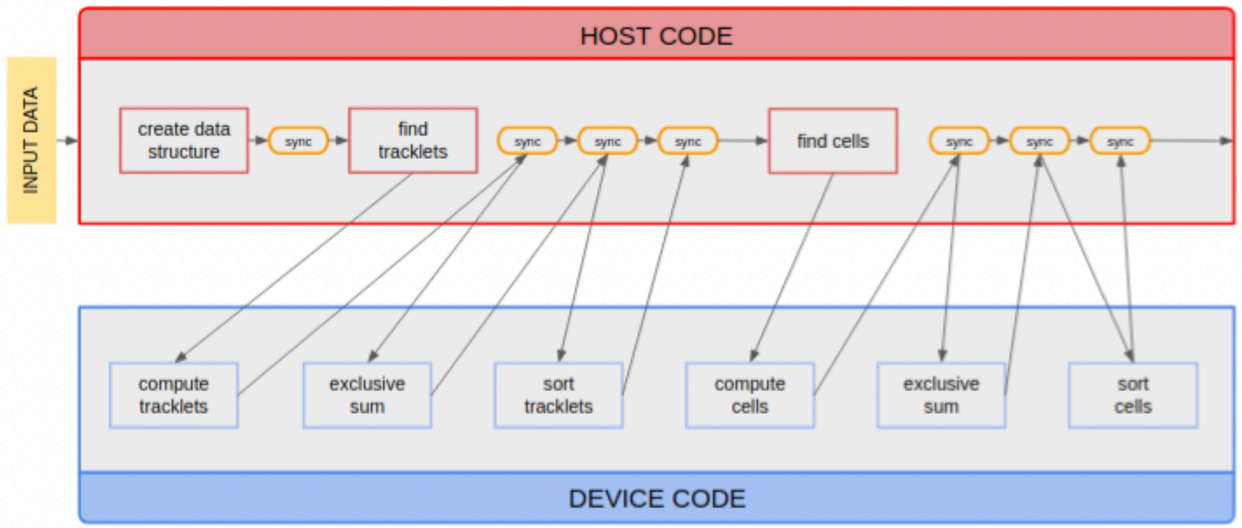
Track Fitting

Neighbour cells are combined into track candidates a fit is later performed using a Kalman Filter

USE-CASE EXAMPLE: CUDA VS OPENCL IMPLEMENTATION

- For each pair of layers an instance of the ITS the trackleting kernel is launched
  - For each cluster in the innermost layer a single thread performs the search for a good tracklet
  - The same strategy and algorithm are used for the cell finding, where tracklets are combined instead of clusters

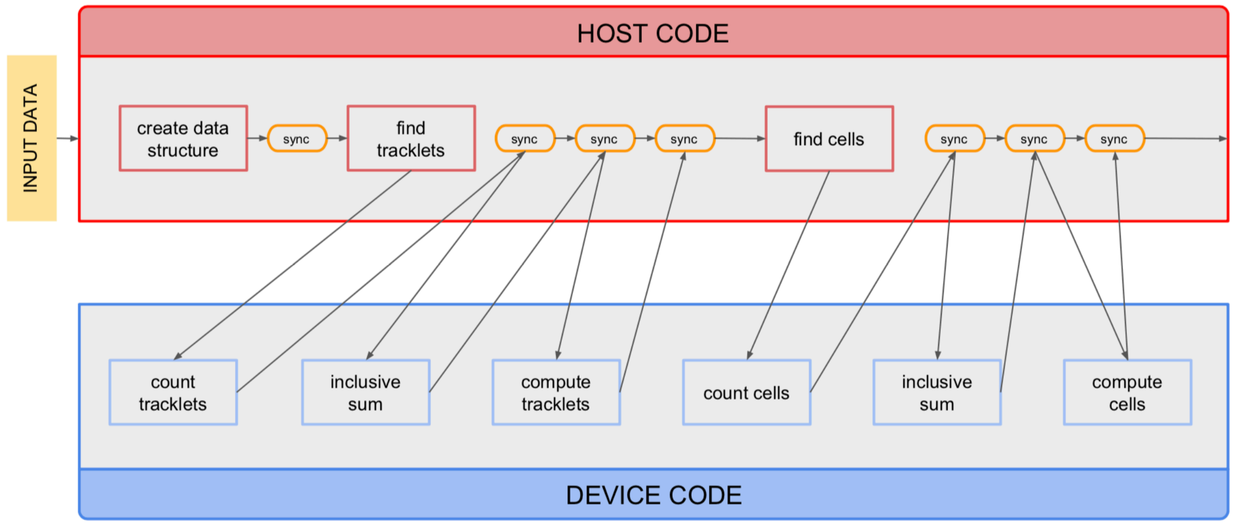
CUDA



Host-device paradigm: host execution, red filled, and device execution, blue filled, must be properly synchronised

- The algorithm has been modified to avoid the sorting of the tracklets and atomic operations
  - A “dry run” of the tracklets/cells finding algorithm is performed in order to count the total number of tracklets/ cells reconstructed per cluster
  - A second iteration of the algorithm is used to instantiate the object (tracklet or cell) in memory already sorted for the following step

OpenCL



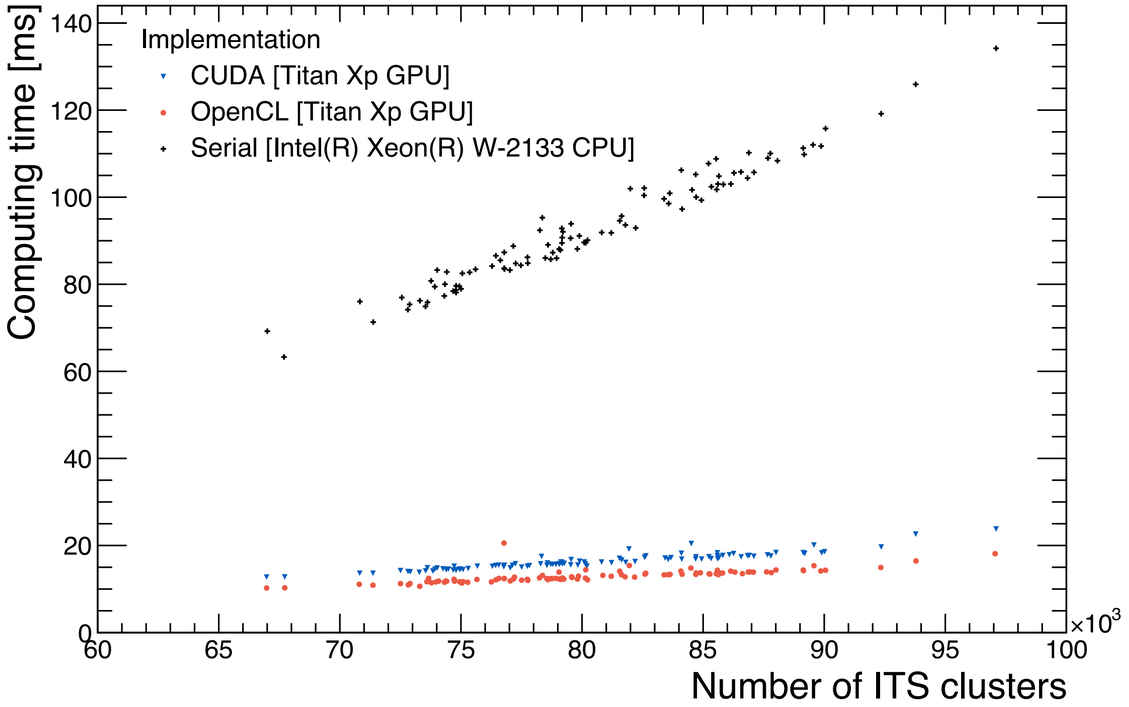
Host-device paradigm: host execution, red filled, and device execution, blue filled, must be properly synchronised

CUDA VS OPENCL IMPLEMENTATION: RESULTS

Test server configuration

- **CPU**: Intel(R) Xeon(R) W-2133 @ 3.60GHz
- **RAM**: 16GB 2666MHz DDR4 ECC RDIMM
- **GPU**: NVIDIA TITAN Xp 12 GB GDDRX5 1405 MHz
- **OS**: Ubuntu 18.04 LTS
  - `gcc 7.3.0` with flag `-O3`
- **OpenCL**: Version 1.2
- **CUDA**: Version 9.2.88

- The algorithm has been tested on central Pb-Pb (HIJING[7]) events simulated using the realistic geometry of the upgraded ITS
- The computing time is reported for the reconstruction of tracks coming from a single interaction vertex
  - Piling up more interaction vertices the computing time increases linearly (see table)
- The OpenCL algorithm is slightly more performant than the CUDA one, both leads to the same results and are consistent with CPU version
- Both GPU implementations show a similar linear dependence on the number of clusters lower than the serial one

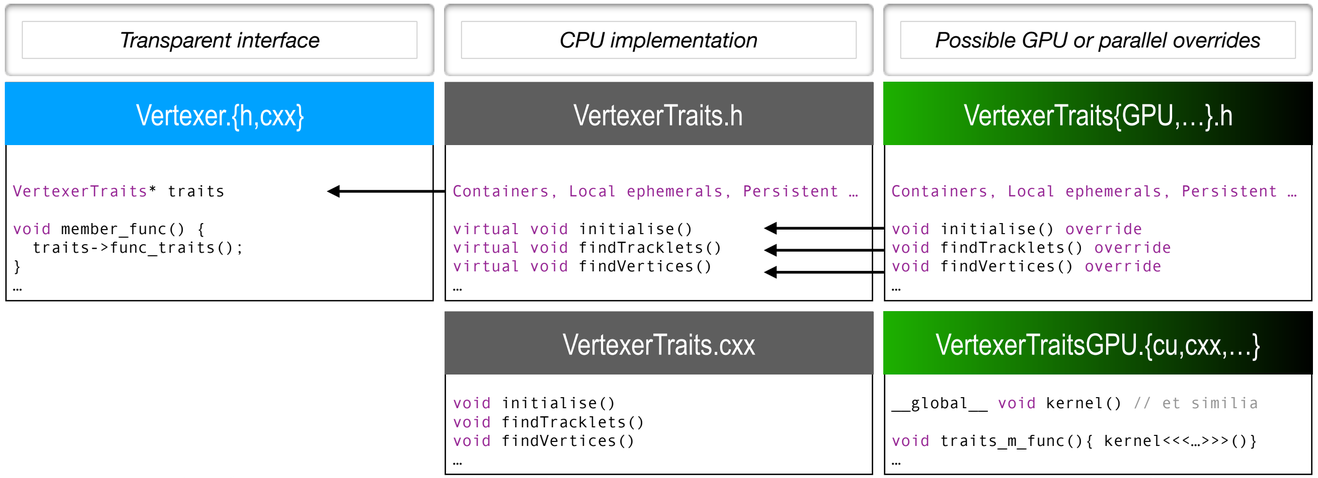


Number of vertices		1			2			4			5		
Context init	min	5.7	4.9	4.5	9.3	8.6	7.6	22.2	20.7	17.9	28.5	27.4	24.3
	avg	7.2	6.6	5.7	13.2	11.8	10.0	28.8	25.8	22.5	38.2	34.1	29.4
	max	10.4	11.5	7.6	19.4	14.4	14.3	49.8	30.7	35.5	66.7	42.0	45.9
Tracklet finding	min	1.7	1.0	23.6	4.2	1.8	102.8	20.0	5.2	435.3	32.3	8.2	93.6
	avg	2.3	1.4	51.6	5.2	2.9	180.2	23.9	7.4	677.3	36.9	10.8	1027.6
	max	3.1	2.0	74.1	7.7	3.7	227.2	29.6	13.0	831.1	44.2	14.2	1259.6
Cell finding	min	1.4	1.0	11.5	4.9	3.7	58.1	31.7	14.0	399.6	50.1	22.7	745.1
	avg	3.7	2.5	31.8	13.3	7.6	151.3	62.4	27.7	943.7	102.9	45.8	1698.1
	max	5.8	3.8	49.5	18.1	10.0	211.0	84.0	37.0	1297.7	139.8	62.9	2324.1
Total	min	9.3	7.8	40.0	19.5	15.9	169.8	78.3	43.0	855.4	117.8	62.5	1466.6
	avg	16.2	12.9	92.0	37.0	26.6	348.2	124.7	69.8	1652.9	191.1	103.0	2768.0
	max	23.9	20.5	134.7	48.1	31.6	455.4	173.1	86.2	2149.3	259.1	133.8	3602.3

Computing time [ms] for CUDA, OpenCL and Serial implementations

PORTABILITY ON HETEROGENEOUS ARCHITECTURES: EXAMPLE OF A TRANSPARENT INTERFACE

- The O^2 software stack will run online and offline, the same code must be able to adapt to the underlying architecture (online clusters, Grid sites)
- There are several strategies. For instance the TPC parallel code is replicated in three different flavours (CUDA, OCL, OpenMP)
- The basic idea is to have transparent interfaces, which implement standard APIs for workflow
  - Interfaces are overridden, the idea is to always choose to use the fastest version available for final architecture
- At the moment, for CUDA, we are able to autodetect the underlying architecture enabling the compilation of the proper piece of code. We would like soon move towards the same direction for HIP and OpenCL



Example of a transparent interface of a CUDA implementation for the ITS seed vertex finder

ALIDOCK: THE ALICE ENVIRONMENT IN A CONTAINER

- The alidock[9] script has been developed to solve a real problem at the ALICE analysis tutorial: make ALICE newcomers able to install, develop and use our experiment codebase without spending too much time in struggling with compatibility*
- Available for Linux and Mac (Windows version has not released yet, CUDA will not be available also with WSL2)
- Focused on simplicity:
  - installation with a single command and minimal CLI with single command for basic usage
  - automatic update both of the executable and the container image (it comes for free with docker)
  - unburden the final user from docker technicalities as much as possible
- The goal: provide users with a consistent environment, based on the upstream production docker container images used in the ALICE software validation → pre-compiled and cached builds for packets not in development mode exist: trade compilation time with downloads (usually faster on users' laptops)

*In ALICE the supported platforms for users are well defined and maintained, the context of an analysis tutorial might find people with exotic environments on their laptops, the idea was not spend too much time in technicalities in fixing different unsupported OSs

ALIDOCK: HOW IT WORKS

- Installation is self contained into a Python 3 virtualenv, to avoid Python prerequisite conflicts
- Possible customisations stored in a static file and overridable by CLI
- Initialisation script runs inside the container at startup to customise execution (user creation, ssh key pairs deployment, specific flags implementations…)
- Expose a default-created directory to store persistent data (configurable)
- Access through the simple `alidock` command (SSH behind the scene)
- Container is meant to be disposable, user should be able to just stop and restart it without noticing any difference (sometimes even useful as a panacea-fix for specific issues)
- It provides a devel and runtime environment for final user
- It does not have a `--privileged` option

ALIDOCK AS AN ADVANCED-USER DEVELOPMENT TOOL

- With the user becoming more familiar with the ALICE development workflow, there may come out more advanced needs
  - preserve the state within a tmux session, able to run things in “background”: `--tmux[-control]`
  - access host directories, CVMFS, cernbox: `--cvmfs, --mount`
  - access the host devices Nvidia or AMD GPUs: `alidock --nvidia / --rocm`
  - exceptionally access the root user: `alidock --root`
- It is possible to derive a custom image from the original one and use it as the base image for your workbench, alidock is completely application-agnostic
- Repository for *contribs* images with automatic build and test exists, publish on dockerhub

CONCLUSIONS AND OUTLOOK

- Having parallel code and GPU accelerators utilisation is a fact for the upgrade. The O^2 framework is intrinsically multi process for different task communicating via the a communication abstraction and allows for multithreaded executions on CPUs and GPUs per single *device*
- Having the same software stack for online and offline data processing will reduce the code duplication and ease further developments and maintenance. Also the goal is to keep interfaces homogeneous and transparent wrt the underlying running implementation → need for continuous consistency testing
- At the moment we are trying not to be vendor locked in for what concern accelerators' code, exploring the most usable, reliable and performant technologies for cross compatibility, looking for being compatible to all the foreseeable scenarios also on grid sites for asynchronous phase
- The most basic scenario where we will have separated workflow on GPU is not so far, we are carefully evaluating which steps would really benefit from being connected in pipelines
- Tools like alidock may enable both the basic and the expert user to an agile development environment, up to date with latest technologies frontiers

BACKUP SLIDES

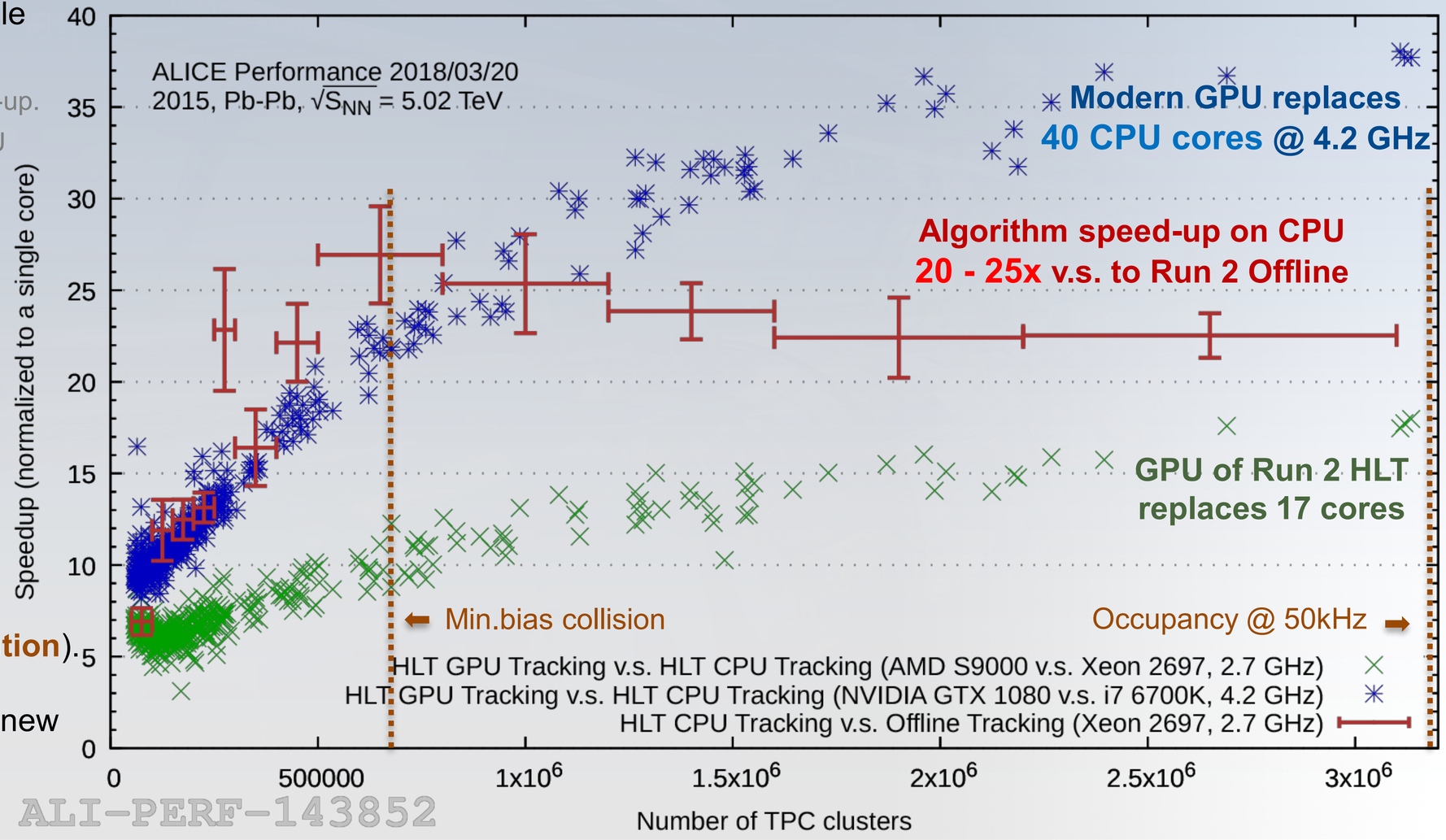
TPC Tracking performance

- Speed-up normalized to single CPU core.
  - Red curve: algorithm speed-up.
  - Other curves: GPU v.s. CPU speed-up corrected for CPU resources.
    - How many cores does the GPU replace.

- Significant gain with newer GPU (blue v.s. green).

- GPU with Run 3 algorithm replaces **> 800 CPU cores** Running Run 2 algorithm. (**blue * red**). (**at same efficiency / resolution**).

- We see ~**30%** speedup with new GPU generation (RTX 2080 v.s. GTX 1080)



D. Rohr - "Track Reconstruction in the ALICE TPC using GPUs for LHC Run 3" Connect the Dots 2019"[6]

REFERENCES

- [1] https://indico.cern.ch/event/587955/contributions/2935765/attachments/1678513/2699330/2018-jul-03-conference_presentation-chep2018-v2.pdf
- [2] https://indico.cern.ch/event/658267/contributions/2813689/attachments/1621144/2579443/2018-03-21_CTD_2018.pdf
- [3] https://developer.nvidia.com/cuda-zone
- [4] https://www.khronos.org/opencl/
- [5] https://gpuopen.com/compute-product/hip-convert-cuda-to-portable-c-code/
- [6] https://indico.cern.ch/event/742793/contributions/3274344/attachments/1823598/2983651/2019-04-04_CTD_2019.pdf
- [7] https://doi.org/10.1103/PhysRevD.44.3501
- [8] https://indico.cern.ch/event/587955/contributions/2935788/attachments/1683802/2706959/mrichter_CHEP2018-alice-o2-epn-processing_final.pdf
- [9] https://github.com/alidock/alidock