Improvements of the GPU Processing Framework for ALICE

David Rohr for the ALICE Collaboration

CHEP 2024

23.10.2024

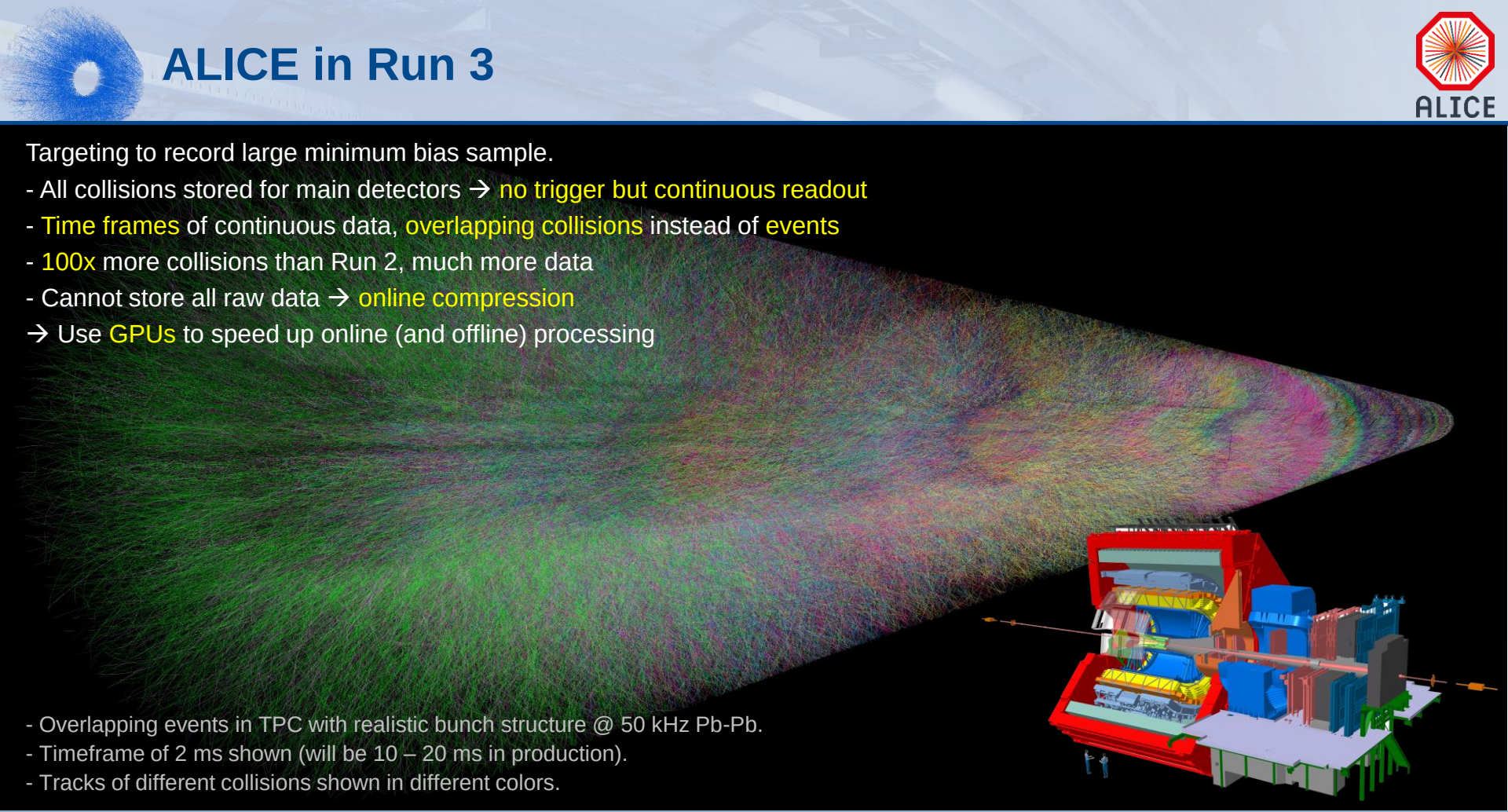
drohr@cern.ch

ALICE in Run 3

Targeting to record large minimum bias sample.

- All collisions stored for main detectors → no trigger but continuous readout
- Time frames of continuous data, overlapping collisions instead of events
- 100x more collisions than Run 2, much more data
- Cannot store all raw data → online compression

→ Use GPUs to speed up online (and offline) processing



- Overlapping events in TPC with realistic bunch structure @ 50 kHz Pb-Pb.
- Timeframe of 2 ms shown (will be 10 – 20 ms in production).
- Tracks of different collisions shown in different colors.

ALICE in Run 3

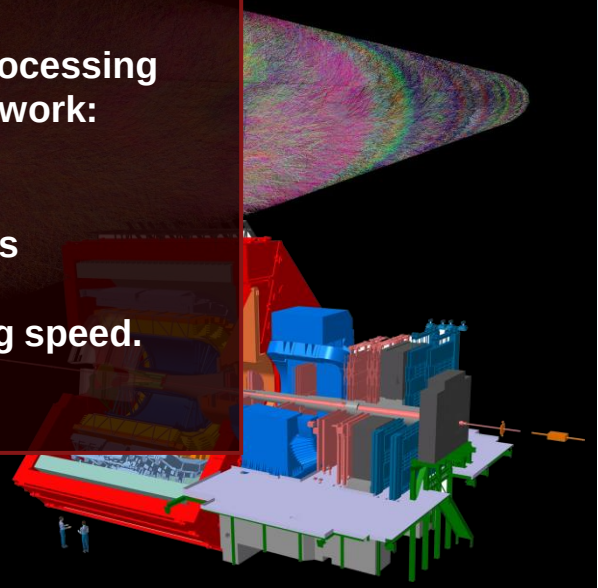
Targeting to record large minimum bias sample.

- All collisions stored for main detectors → no trigger but continuous readout
- Time frames of continuous data, overlapping collisions instead of events
- 100x more collisions than Run 2, much more data
- Cannot store all raw data → online compression

→ Use GPUs to speed up online (and offline) processing

Outline:

- **Fast executive summary of ALICE GPU processing**
- **Explain new features of ALICE GPU framework:**
  - **Deterministic Mode**
  - **Per-kernel compilation**
  - **Object library for shared components**
  - **Run Time Compilation**
- **Evolution of ALICE GPU online processing speed.**



- Overlapping events in TPC with realistic bunch structure @ 50 kHz Pb-Pb.
- Timeframe of 2 ms shown (will be 10 – 20 ms in production).
- Tracks of different collisions shown in different colors.

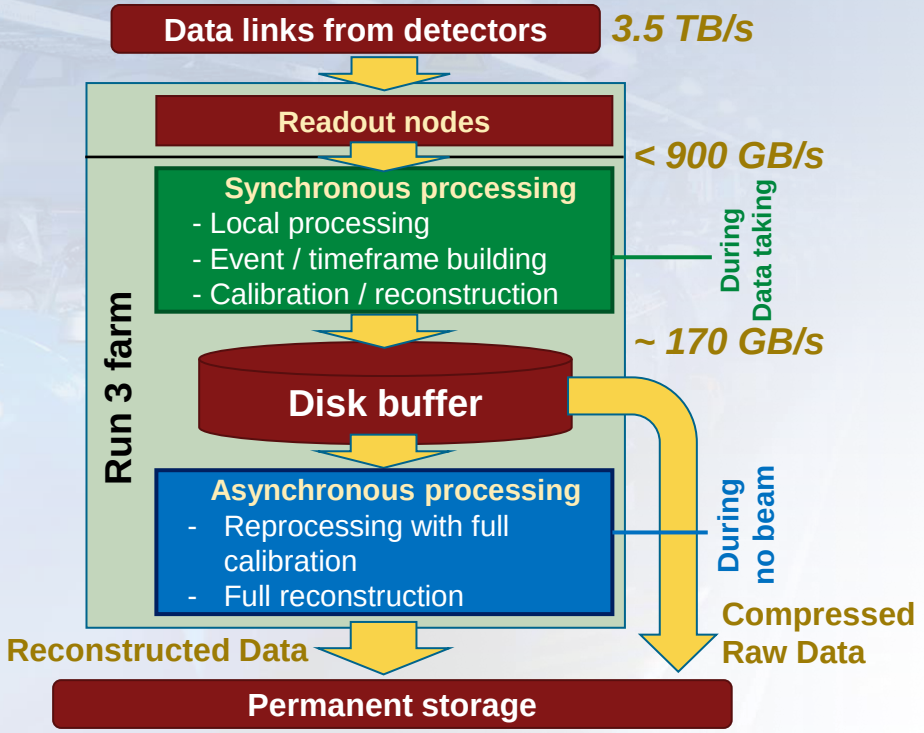
Executive summary: ALICE online / offline processing

- **During LHC operation:**
  - **Online processing** on **GPUs.**
  - **Compressed raw data** stored to **disk buffer**.
- **When no beam in LHC:**
  - **Offline processing** data from disk buffer on **online farm**.
  - **Offline processing** always running in the **GRID**.
  - **Optionally** use **GPUs** when possible.

- **Online computing farm of 350 servers, 2800 GPUs.**

- **Baseline scenario:**
  - All that is **mandatory** for **online processing:** **Full TPC processing on GPU.**
- **Optimistic scenario:**
  - Try to **offload more algorithms** to **GPU** for **better GPU usage in offline**.

See **talk** of last CHEP



Executive summary: GPU usage and speedup

Online reconstruction
(50 kHz Pb-Pb, MC data, no QA / calib)

Processing step	% of time
TPC Processing	99.37 %
EMCAL Processing	0.20 %
ITS Processing	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Baseline scenario (today): Online processing totally dominated by TPC on GPU

Offline processing
(650 kHz pp, 2022, no Calorimeters)

Processing step	% of time
TPC Processing	61.41 %
ITS TPC Matching	6.13 %
MCH	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Baseline scenario (today): ~60% on GPU → 2.5x speedup in offline

Offline processing
(47 kHz Pb-Pb, 2024)

Processing step	% of time
TPC Processing	52.39 %
ITS Tracking	12.65 %
Secondary Vertexing	8.97 %
MCH	5.28 %
TRD Tracking	4.39 %
TOF Matching	2.85 %
ITS TPC Matching	2.64 %
Entropy Decoding	2.63 %
AOD Production	1.72 %
Quality Control	1.64 %
Rest	4.84 %

Executive summary: GPU usage and speedup

Online reconstruction
(50 kHz Pb-Pb, MC data, no QA / calib)

Processing step	% of time
TPC Processing	99.37 %
EMCAL Processing	0.20 %
ITS Processing	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Baseline scenario (today): Online processing totally dominated by TPC on GPU

- **One AMD MI50 GPU replaces 80 CPU cores.**
- **Hypothetical CPU-only online farm would require more than 3000 64-core servers.**
- **GPU usage mandatory for ALICE in Run 3.**

Offline processing
(650 kHz pp, 2022, no Calorimeters)

Processing step	% of time
TPC Processing	61.41 %
ITS TPC Matching	6.13 %
MCH	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Baseline scenario (today): ~60% on GPU → 2.5x speedup in offline

Offline processing
(47 kHz Pb-Pb, 2024)

Processing step	% of time
TPC Processing	52.39 %
ITS Tracking	12.65 %
Secondary Vertexing	8.97 %
MCH	5.28 %
TRD Tracking	4.39 %
TOF Matching	2.85 %
ITS TPC Matching	2.64 %
Entropy Decoding	2.63 %
AOD Production	1.72 %
Quality Control	1.64 %
Rest	4.84 %

Offline reco on GPU server

Configuration (2022 pp, 650 kHz)	Time per TF (7400 collisions)
8 * 16-core CPU workflow	4.27s
2 * 1 NUMA domain (4 GPUs + 64 cores)	1.70s

Factor 2.51

Executive summary: GPU usage and speedup

Online reconstruction
(50 kHz Pb-Pb, MC data, no QA / calib)

Processing step	% of time
TPC Processing	99.37 %
EMCAL Processing	0.20 %
ITS Processing	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Baseline scenario (today): Online processing totally dominated by TPC on GPU

Offline processing
(650 kHz pp, 2022, no Calorimeters)

Processing step	% of time
TPC Processing	61.41 %
ITS TPC Matching	6.13 %
MCH	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Baseline scenario (today): ~60% on GPU → 2.5x speedup in offline

Offline processing
(47 kHz Pb-Pb, 2024)

Processing step	% of time
TPC Processing	52.39 %
ITS Tracking	12.65 %
Secondary Vertexing	8.97 %
MCH	5.28 %
TRD Tracking	4.39 %
TOF Matching	2.85 %
ITS TPC Matching	2.64 %
Entropy Decoding	2.63 %
AOD Production	1.72 %
Quality Control	1.64 %
Rest	4.84 %

Optimistic scenario (future): ~80% on GPU → 5x expected in offline

- **One AMD MI50 GPU replaces 80 CPU cores.**
- **Hypothetical CPU-only online farm would require more than 3000 64-core servers.**
- **GPU usage mandatory for ALICE in Run 3.**

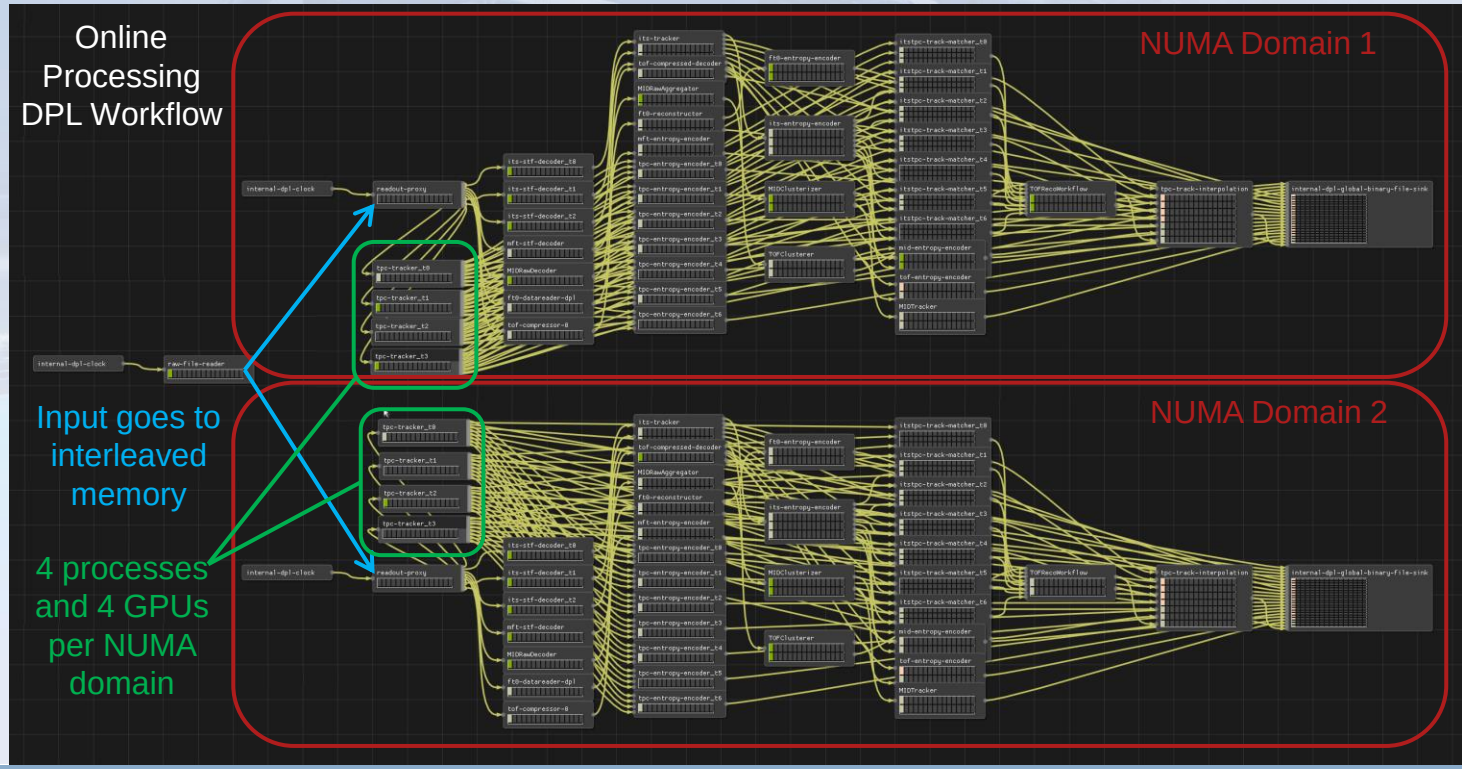
Offline reco on GPU server

Configuration (2022 pp, 650 kHz)	Time per TF (7400 collisions)
8 * 16-core CPU workflow	4.27s
2 * 1 NUMA domain (4 GPUs + 64 cores)	1.70s

Factor 2.51

Executive summary: Experience running with GPUs

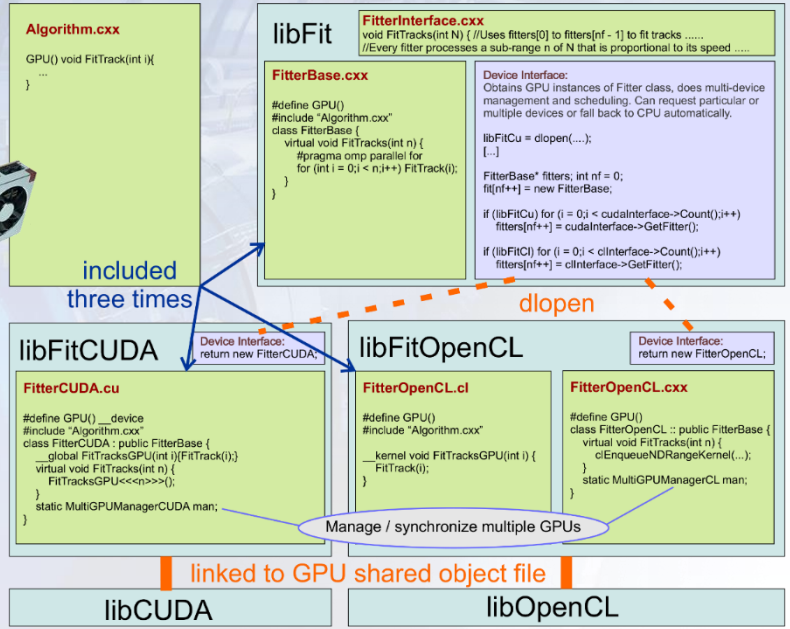
- **ALICE was running the GPU-enabled online workflow successfully for pp and Pb-Pb from 2022 to 2024.**
  - During **2023 Pb-Pb** had **17% free GPU resources** at highest interaction rate.



Executive summary: Experience running with GPUs

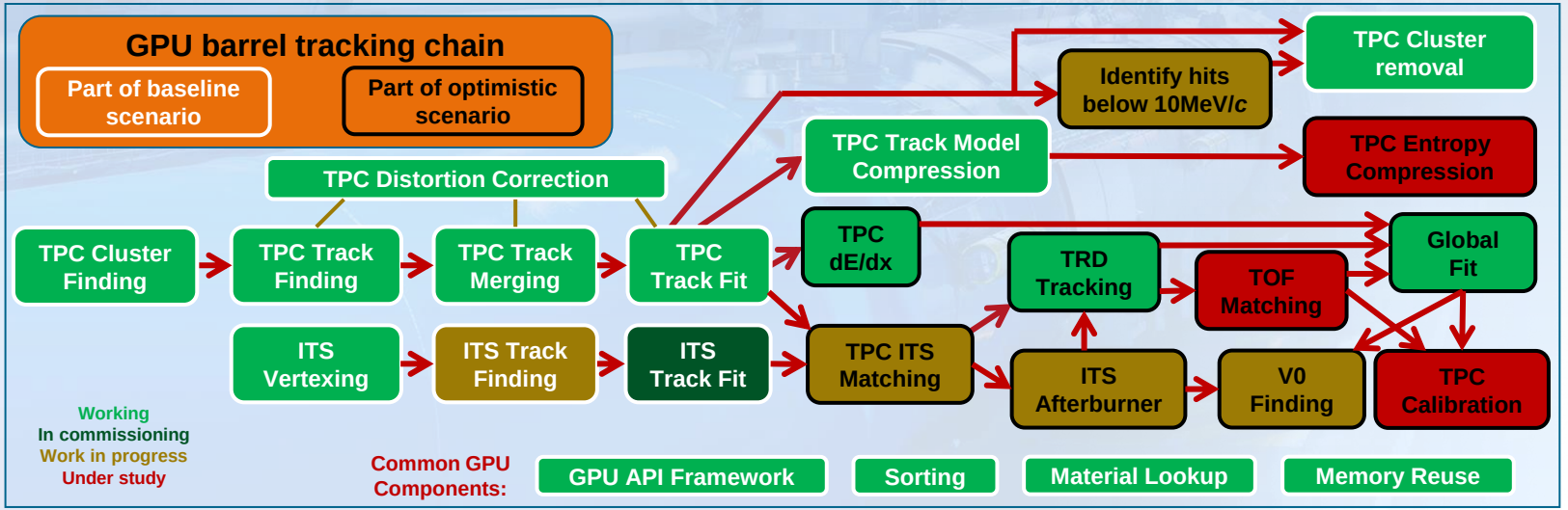
- **ALICE was running the GPU-enabled online workflow successfully for pp and Pb-Pb from 2022 to 2024.**
  - During 2023 Pb-Pb had 17% free GPU resources at highest interaction rate.
- **ALICE does vendor-independent GPU usage via generic common C++ Code.**
  - Can run on CUDA, OpenCL, HIP, and CPU (with pure C++, OpenMP, or OpenCL).

- CPUs (AMD Zen, Intel Skylake)
  C++ backend with OpenMP, AMD OCL
- AMD GPUs
  (S9000 with OpenCL 1.2, MI50 / Radeon 7 / Navi with HIP / OCL 2.x)
- NVIDIA GPUs
  (RTX 2080 / RTX 2080 Ti / Tesla T4 with CUDA)
- ARM Mali GPU with OCL 2.x
  (Tested on dev-board with Mali G52)



Executive summary: Experience running with GPUs

- **ALICE was running the GPU-enabled online workflow successfully for pp and Pb-Pb from 2022 to 2024.**
  - During 2023 Pb-Pb had 17% free GPU resources at highest interaction rate.
- **ALICE does vendor-independent GPU usage via generic common C++ Code.**
  - Can run on CUDA, OpenCL, HIP, and CPU (with pure C++, OpenMP, or OpenCL).
- **Planning to run full barrel tracking on GPU in optimistic scenario, to raise fraction on GPU from 60% to 80%.**

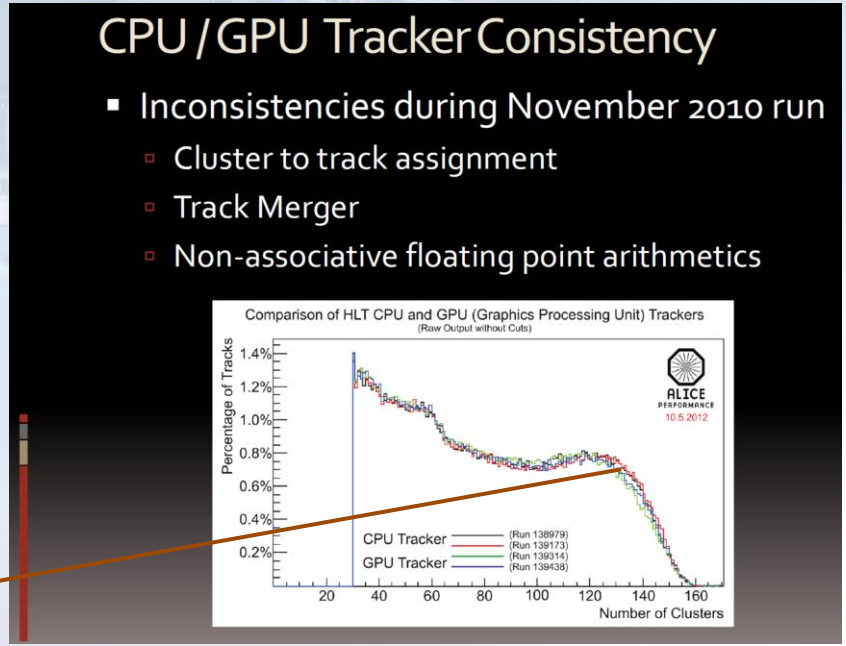


Consistent / reproducible code on GPU and CPU

- **Debugging GPU code / comparing results to CPU difficult if output is not deterministic / differs between CPU and GPU.**
  - So far we always thought:
    - **Non-associative floating point** math from e.g. **–ffast-math** makes the code **indeterministic** anyway.
    - Thus we **accept to anyway have small differences**.
  - We tried to **keep the differences small**, but accepted that **CPU / GPU** output will not be identical.

Presented as such several times
see **CHEP 2012 talk**

Inconsistenst GPU /
CPU results



Consistent / reproducible code on GPU and CPU

- **Debugging GPU code / comparing results to CPU difficult if output is not deterministic / differs between CPU and GPU.**
  - So far we always thought:
    - **Non-associative floating point** math from e.g. **–ffast-math** makes the code **indeterministic** anyway.
    - Thus we **accept to anyway have small differences**.
  - We tried to **keep the differences small**, but accepted that **CPU / GPU** output will not be identical.

- **In 2024, started a new attempt to get it 100% identical and deterministic.**
  - Added "**deterministic mode**": special slow mode for debugging.
    - **Prevents** all **concurrency-caused indeterministic behaviour** and all differences between CPU and GPU.
    - **Combination** of **runtime setting** (where possible) and different **compile-options**.
  - **Quite successful**:
    - **Revealed several bugs** in **our code** and **GPU compiler** (before, assumed to be small deviations due to parallelism).

- **Note:**
  - **Default mode**: **No performance impact at all**.
  - **Deterministic mode**: **Severe performance impact**, **needs recompilation** unfortunately.
    - Quite unlikely case, but does not help for compiler bugs appearing only in normal mode.

Consistent / reproducible code on GPU and CPU

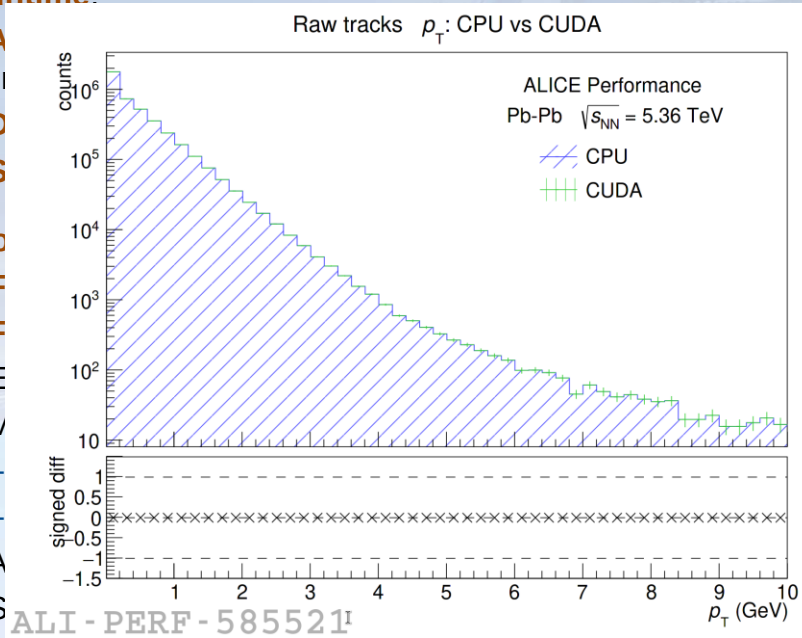
- **Modifications for fully deterministic and identical CPU and GPU results:**
  - **At runtime**:
    - **Additional GPU sorting kernels** (executed after kernels whose order of the output is not deterministic, preserving all links, e.g. cluster to track attachment).
    - **Disable parallel processing** in some cases.
    - **Slower deterministic algorithms** in some cases.

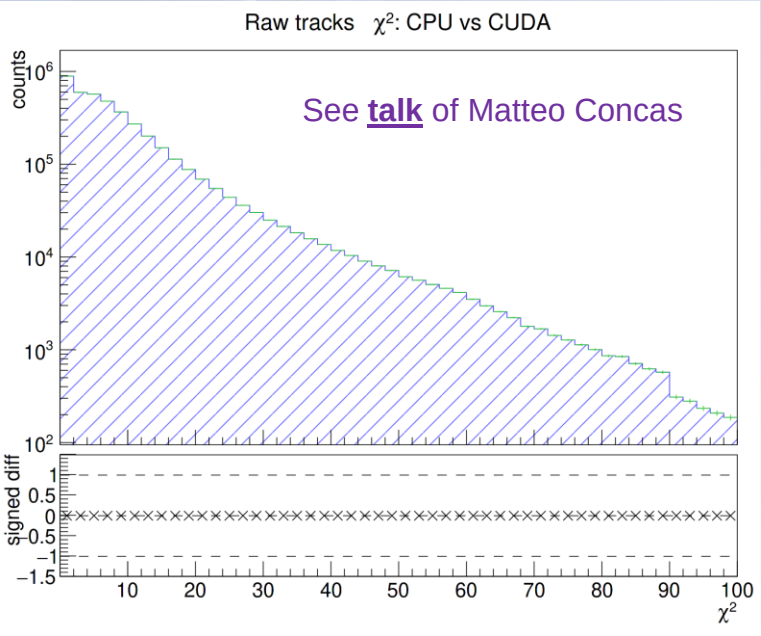
Consistent / reproducible code on GPU and CPU

- **Modifications for fully deterministic and identical CPU and GPU results:**
  - **At runtime**:
    - **Additional GPU sorting kernels** (executed after kernels whose order of the output is not deterministic, preserving all links, e.g. cluster to track attachment).
    - **Disable parallel processing** in some cases.
    - **Slower deterministic algorithms** in some cases.
  - **At compile-time**:
    - **FMA** and ***–ffast-math*** must be **disabled**.
    - **Fast math approximations** like fast inverse square root, etc. are **disabled**.
    - Enforce **consistent behaviour** for **denormalized float** values.
    - Make sure **trigonometric functions** and **other math functions** are **consistent**.
      - **No IEEE float standard requirement for full accuracy**.
      - Only need FP32 for GPU code, so compute in FP64 and round to FP32.
    - All **sorting** must use a **total ordering** (add bytewise sorting on top for partial orders).
    - Some **indeterministic algorithms** are **replaced** by slower deterministic ones.
  - **Performance impact**: **2x** to **10x slower** (mostly due to sorting), **2x memory** requirement.

Consistent / reproducible code on GPU and CPU

- **Modifications for fully deterministic and identical CPU and GPU results:**
  - **At runtime**:
    - **A**
      li
    - **D**
    - **S**
  - **At co**
    - **F**
    - **F**
    - E
    - M
      –
      –
    - A
    - S





- **Today available to all GPU detector code (not only TPC)**.
  - Plot shows **bit-by-bit identical result** of **ITS track fit** comparing **CPU** to **CUDA**.

Per-kernel compilation / per-kernel compile flags

- **Up until 2024, ALICE compiled all GPU code (per backend) in one big file aggregating all GPU kernels.**
  - With > 100 kernels, **compile time** became a **serial bottleneck**, will get worse in the future.

Per-kernel compilation / per-kernel compile flags

- **Up until 2024, ALICE compiled all GPU code (per backend) in one big file aggregating all GPU kernels.**
  - With > 100 kernels, **compile time** became a **serial bottleneck**, will get worse in the future.

- **Attempted CUDA and ROCm RDC (relocatable device code: separate compilation, then link together).**
  - Works, but:
    - GPU compiler **assumes too high number of available registers** for some functions.
    - Must **manually set the register limits (**depending on GPU architecture / in which functions they are used).
    - Observed up to **10% performance degradation**.
  - Thus, decided **not to use RDC** for the majority of the GPU code.
    - Still **available if desired by developers** of detector GPU code.

Per-kernel compilation / per-kernel compile flags

- **Up until 2024, ALICE compiled all GPU code (per backend) in one big file aggregating all GPU kernels.**
  - With > 100 kernels, **compile time** became a **serial bottleneck**, will get worse in the future.

- **Attempted CUDA and ROCm RDC (relocatable device code: separate compilation, then link together).**
  - Works, but:
    - GPU compiler **assumes too high number of available registers** for some functions.
    - Must **manually set the register limits (**depending on GPU architecture / in which functions they are used).
    - Observed up to **10% performance degradation**.
  - Thus, decided **not to use RDC** for the majority of the GPU code.
    - Still **available if desired by developers** of detector GPU code.

- **Implemented framework to compile each kernel in a separate compilation unit using CMake.**
  - List of kernels moved from C++ header to CMake (C++ headers auto-generated).
  - Explicitly calling nvcc and hipcc to compile individual kernels to binary output files.
  - Loaded via ***cuModuleLoad*** and ***hipModuleLoad*** (as for RTC code, see later).
  - **Optimal kernel performance** (compiler can optimize all used functions for the very kernel).
  - **Parallel fast compilation** (measured in wall time, while CPU time actually increases a bit).

Shared components / Constant memory

- **ALICE GPU strategy: provide common framework code, but allow detectors to be as independent as they want.**
  - **May use common features** (e.g. per-kernel compilation).
  - May opt to **compile .cu** and **.hip** files **manually** via CMake.
    - Usually still need **common components** like **geometry**, **propagator**, etc.
    - Common GPU framework components available as **externalProvider CMake object library**.
      - **One limitation**: must **use RDC** (needed for linking multiple GPU objects files).

Shared components / Constant memory

- **ALICE GPU strategy: provide common framework code, but allow detectors to be as independent as they want.**
  - **May use common features** (e.g. per-kernel compilation).
  - May opt to **compile .cu** and **.hip** files **manually** via CMake.
    - Usually still need **common components** like **geometry**, **propagator**, etc.
    - Common GPU framework components available as **externalProvider CMake object library**.
      - **One limitation**: must **use RDC** (needed for linking multiple GPU objects files).

- **One general unsolved problem for ALICE is constant memory.**
  - Common components use **constant memory**.
  - In theory, **one single global instance** could hold all constants.
  - But **CUDA** / **HIP** have **one instance per compilation** unit (non-RDC code) or **one instance per library** (RDC code).
    - **Impossible to use a single GPU memory buffer** for all components.
  - Current ALICE **workaround**:
    - Per-kernel compilation units and externalProviders **register** their **constant memory buffers**.
    - **Transparent for the user** using weak symbols.
    - Constant memory writes **automatically update all instances**.
  - Works, but **huge increase of constant memory writes** (>100 writes to update a single global value).

Run Time Compilation

- **Use runtime configuration knowledge to create faster code.**
  - ALICE uses **Run Time Compilation** (**RTC**) to **recompile device code at runtime** with **optimizations**.
    - Using the **same infrastructure** as for **per-kernel-compilation** (last slide).
    - Source code embedded in GPU library, nvcc / hipcc compile the code at runtime, loaded with ModuleLoad API.
    - Cannot use **CUDA** / **ROCm RTC libraries**, cause **problems** with ***cub*** and ***thrust***.
    - Want to **extend RTC to OpenCL** in the future, and perhaps even to **CPU code** with clang.

Run Time Compilation

- **Use runtime configuration knowledge to create faster code.**
  - ALICE uses **Run Time Compilation** (**RTC**) to **recompile device code at runtime** with **optimizations**.
    - Using the **same infrastructure** as for **per-kernel-compilation** (last slide).
    - Source code embedded in GPU library, nvcc / hipcc compile the code at runtime, loaded with ModuleLoad API.
    - Cannot use **CUDA** / **ROCm RTC libraries**, cause **problems** with ***cub*** and ***thrust***.
    - Want to **extend RTC to OpenCL** in the future, and perhaps even to **CPU code** with clang.
  - Special treatment for **joint multi-core compilation on the online farm**:
    - Online farm has 8 GPUs, driven by 8 different OS processes.
    - **RTC** code **compiled once**, **stored in cache**, and **reused** (if same settings), synchronized with file locks.
    - **CPU affinity pinning** must be **released** to use all cores for compilation.

Run Time Compilation

- **Use runtime configuration knowledge to create faster code.**
  - ALICE uses **Run Time Compilation** (**RTC**) to **recompile device code at runtime** with **optimizations**.
    - Using the **same infrastructure** as for **per-kernel-compilation** (last slide).
    - Source code embedded in GPU library, nvcc / hipcc compile the code at runtime, loaded with ModuleLoad API.
    - Cannot use **CUDA** / **ROCm RTC libraries**, cause **problems** with ***cub*** and ***thrust***.
    - Want to **extend RTC to OpenCL** in the future, and perhaps even to **CPU code** with clang.
  - Special treatment for **joint multi-core compilation on the online farm**:
    - Online farm has 8 GPUs, driven by 8 different OS processes.
    - **RTC** code **compiled once**, **stored in cache**, and **reused** (if same settings), synchronized with file locks.
    - **CPU affinity pinning** must be **released** to use all cores for compilation.
  - **Optimizations** currently used for RTC:
    - **Configuration parameters constant during a run** are **replaced** with ***constexpr*** constants.
      - Allows the compiler to **optimize away a lot of unused code** and **reduce number of branches**.
    - **Code used only in offline** processing is **removed** in online to reduce branches (hidden by #ifdef).

Run Time Compilation

- **Use runtime configuration knowledge to create faster code.**
  - ALICE uses **Run Time Compilation** (**RTC**) to **recompile device code at runtime** with **optimizations**.
    - Using the **same infrastructure** as for **per-kernel-compilation** (last slide).
    - Source code embedded in GPU library, nvcc / hipcc compile the code at runtime, loaded with ModuleLoad API.
    - Cannot use **CUDA** / **ROCm RTC libraries**, cause **problems** with ***cub*** and ***thrust***.
    - Want to **extend RTC to OpenCL** in the future, and perhaps even to **CPU code** with clang.
  - Special treatment for **joint multi-core compilation on the online farm**:
    - Online farm has 8 GPUs, driven by 8 different OS processes.
    - **RTC** code **compiled once**, **stored in cache**, and **reused** (if same settings), synchronized with file locks.
    - **CPU affinity pinning** must be **released** to use all cores for compilation.
  - **Optimizations** currently used for RTC:
    - **Configuration parameters constant during a run** are **replaced** with ***constexpr*** constants.
      - Allows the compiler to **optimize away a lot of unused code** and **reduce number of branches**.
    - **Code used only in offline** processing is **removed** in online to reduce branches (hidden by #ifdef).
  - Helps to **provide software release** for **larger set** of **device architectures**:
    - ALICE **CVMFS builds** currently contain only device code for **1 NVIDIA and 2 AMD GPU models**.
    - With **RTC**, device code for the **current architecture** can be **created on the fly**, without blowing up CVMFS build time.

TPC Processing speed evolution since 2023 Pb-Pb

- **ALICE GPU code still under heavy development.**
  - **New features and code changes** caused aggregate **slowdown of 43.8%** in 2024
    - Some of the code only needed for offline but affecting online performance.
  - **New features** (e.g. RTC) and **code optimization** could regain most of it, **aiming for 9% faster code** than 2023 eventually.

Software version	Time per TF	Performance impact	Performance vs 2023 Pb-Pb
2023 Pb-Pb software version	3.430 s	0.0 %	0.0 %
Build average q, use q in cluster errors	3.832 s	-11.7 %	-11.7 %
Dead channel map + V/M shape correction map	4.678 s	-24.6 %	-36.4 %
Unrelated change in MatLUT causing ROCm compiler problem	4.812 s	-3.9 %	-40.3 %
IFC cluster errors	4.898 s	-2.5 %	-42.8 %
Other changes	**4.934 s**	**-1.1 %**	**-43.8 %**
Code improvements, fix MatLUT problem, better compile flags	4.756 s	+5.2 %	-38.6 %
RTC constexpr optimization	4.572 s	+5.4 %	-33.3 %
Today: RTC mitigation for V/M shape corrections	**3.875 s**	**+20.3 %**	**-13.0 %**
In the future: merge TPC corrections in single transform map	3.126 s	+21.8 %	+8.9 %

- *Note that this table sums up effects of individual commits related to a topic, not following the real chronological order of commits, thus ignoring all interplay of the commits. This is not fully precise but gives an overview.*
- *All speedups / slowdowns here are normalized to the 2023 Pb-Pb time. Measuring step by step would give other percentages.*
- *Measured on a reference MC time frame with 100 Pb-Pb collisions on an online compute node with AMD MI50 GPU.*

Conclusions

- **ALICE employs GPUs heavily to speed up online and offline processing.**
  - **99%** of **synchronous reconstruction** on the **GPU** (no reason at all to port the rest).
    - **Smooth online operation** from 2022 to 2024.
  - Today ~**60%** of full **asynchronous processing** (for 650 kHz pp) on **GPU** yielding **2.5x speedup** on GPU farm.
    - Will increase to **80%** with full barrel tracking (**optimistic scenario**), aiming for **5x speedup**.

- **Continuous development of GPU code.**
  - **New features need additional processing time**, but **code is also continuously improved**, aiming for **10% faster code next year**.

- **Several new ALICE GPU framework features available:**
  - **Per-kernel compilation** improves compile time.
  - **Run-Time-Compilation RTC** enables additional optimization, and creates device code for additional architectures on the fly.
  - **Deterministic mode** helps debugging and validation, revealed several code and compiler bugs.
  - **Relocatable device code** makes common components available to external libraries where needed.