GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Bundesministerium für Bildung und Forschung

CBM

Federated Digital Infrastructures for Research on Universe and Matter

Optimization of compute-intensive reconstruction tasks for efficient usage of CPUs and GPUs

G.Kozlov, A.Redelbach

30.09.2024

Introduction

- CBM — future fixed-target heavy-ion experiment at FAIR, Darmstadt, Germany.
- 2.5A to 11A GeV Au+Au collisions.
- Interaction rate: 10^5-10^7 collisions per second.
- Up to 1000 charged particles/collision.
- Free streaming data, no hardware triggers.
- **Online time-based event reconstruction** and selection is required in the first trigger level.

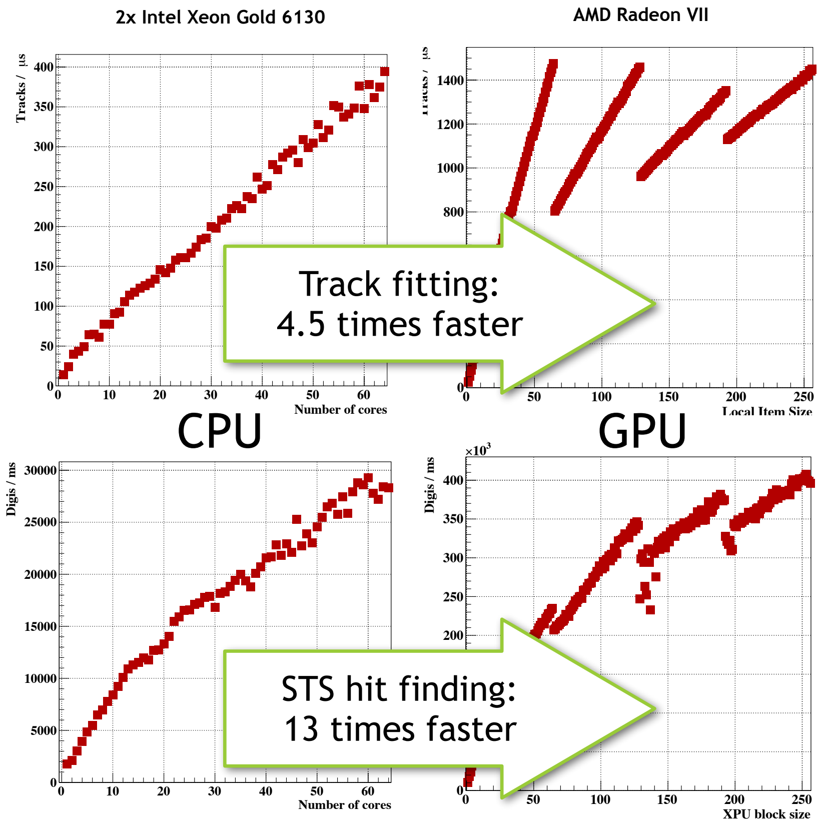
Computing resources:

- GSI Green IT Cube
    - ~54000 CPU cores;
    - ~400 GPUs;
    - PUE: < 1.07
- Goethe NHR
    - ~30000 CPU cores;
    - 864 GPUs;
    - PUE: 1.067

Center for Scientific Computing Frankfurt

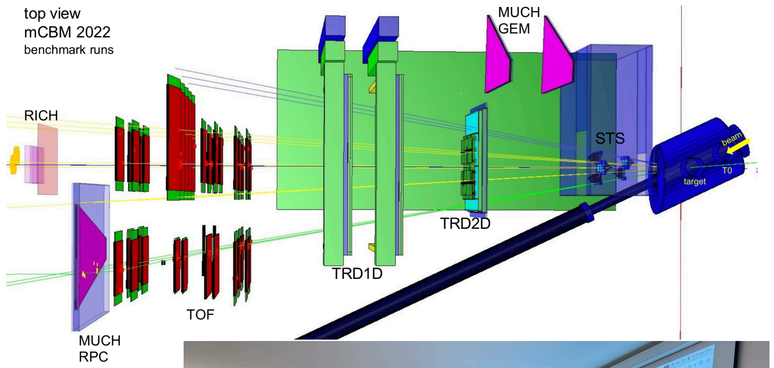
CPU vs GPU: previous studies

- CPUs are more flexible while running of complicated algorithms.
- GPUs are much faster while doing large amount of similar calculations.
- Previous studies demonstrate significant superiority of GPUs in solving tasks such as hit detection and track fitting.
- Reconstruction in the CBM experiment still tends to use CPU algorithms.
- More active utilization of GPUs is encouraged where it can be useful.



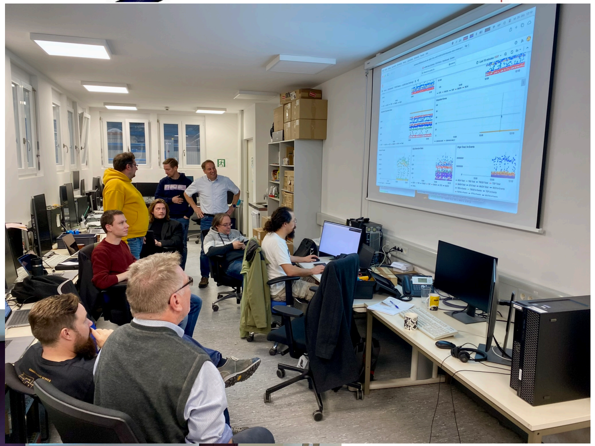
- Transferring massive computing tasks to the GPU will allow for fast processing of large amounts of data.
- CPU usage can be focused on tasks that are fundamentally impossible or unprofitable on the GPU, thus optimizing the use of computing resources.

CBM data challenges using prototype mCBM setup



Motivation:
- Ensure interoperability
- Identify and address missing components
- Focus on actually employable software
- Reasonable milestones and deadlines
- Make developments visible

Concept for setup:
- Emulate the actual online processing as closely as possible
- Replay recorded data (due to lack of synthetic data)
- mFLES nodes sending timeslice data to be processed by GSI-VIRGO HPC nodes



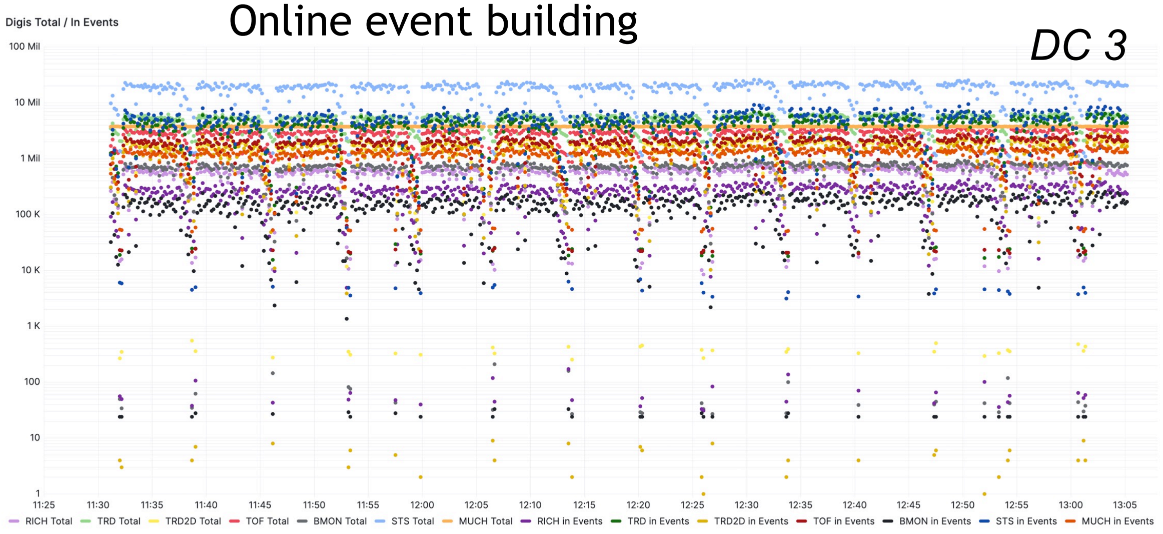
- Execute data chain using recorded data
- Container operation: binaries built into Docker/Apptainer container
- 4 nodes sending recorded raw data (timeslice format)
- Parallel operation of 10+ processing nodes
- On node parallelization: multi process plus OpenMP

CBM data challenges

- Unpacking and event building with digi triggering of mCBM detector data;
- Unpackers for all detectors are included;
- All parts of the integrated processing chain work stably;
- Proper operation of container based solution on GSI VIRGO-HPC cluster.

Online event building

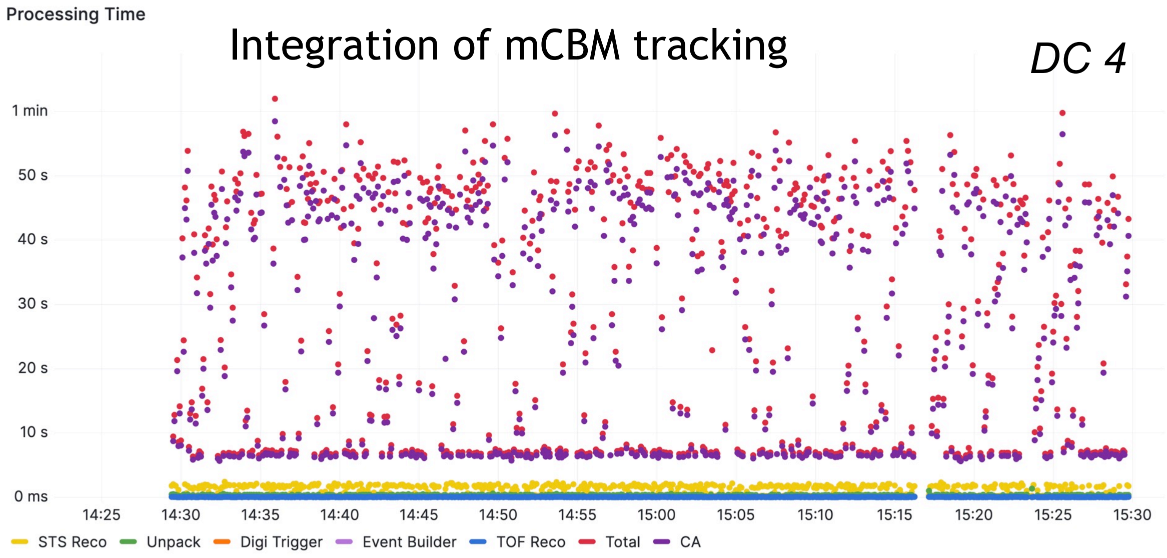
DC 3



- Chain includes unpacking, local reconstruction, CA track finder and writing of the output data;
- Tracking is based on STS and TOF hits;
- **CA tracking dominates the total runtime.**

Integration of mCBM tracking

DC 4



Event reconstruction on GPU

Online event reconstruction:

- Algorithms:
  - Hit finding - STS detector - CPU/GPU;
  - Track reconstruction - CA tracker - CPU, **GPU in progress**;
- Running:
  - Customizable scripts for building containers - Docker/Apptainer;
  - Scripts for submitting SLURM jobs: container based on GSI-VIRGO (data challenge), benchmarks on Goethe NHR.

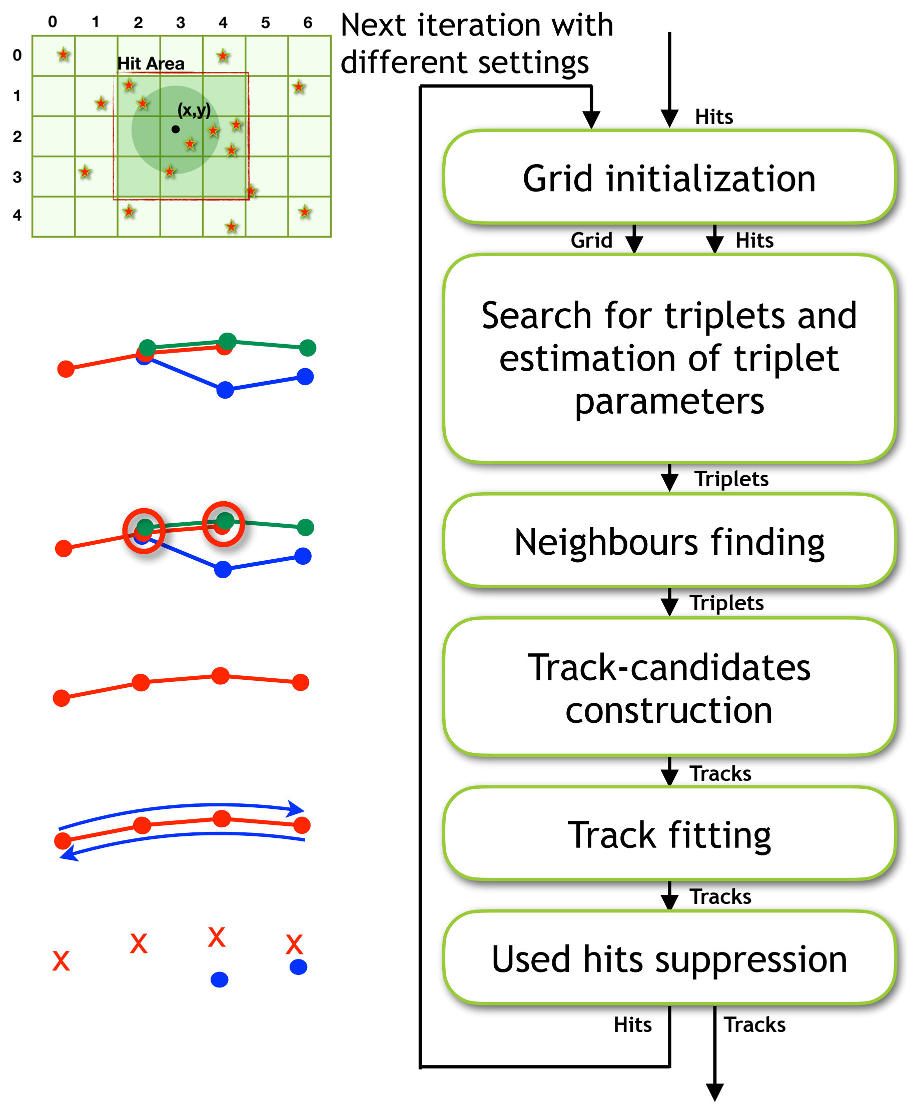
GPU tracking requirements: similar CPU/GPU implementations, high operating speed, proper utilization of CPUs and GPUs, flexibility.

XPU - lightweight C++ library for GPU software development:

- Abstraction layer for specific GPU API/compiler.
- Compiles code as CUDA, HIP or regular C++ with OpenMP.
- Provides optimized GPU algorithms with CPU fallback.
- Collects timing data and manages memory allocation.
- Device code is compiled once for each available backend. Device selection occurs at runtime.

https://github.com/fweig/xpu

Track reconstruction on GPU (under development)



- Simple and fast

- The most resource- and time-consuming part. Combinatorial selection of hit combinations + segment fitting. Large volume of mutually independent calculations.
- CPU parallelization is not very effective.
- There are possibilities for parallel subtractions on the GPU (?)

- Simple and fast

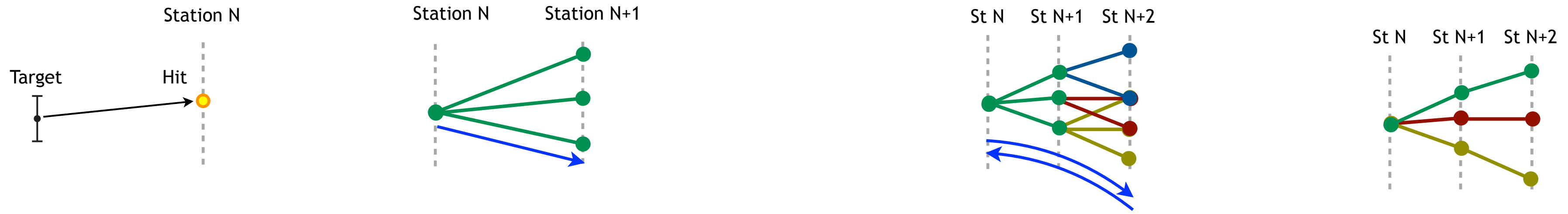
- Recursive function; not parallelizable; for parallelization on GPU it is necessary to rework the function from scratch

- Ready for GPU calculations; will be translated to GPU

Track reconstruction: CPU and GPU

CPU:

FindSinglets() | FindDoublets() | FitDoublets() | FindTriplets() | FitTriplets() | StoreTriplets()



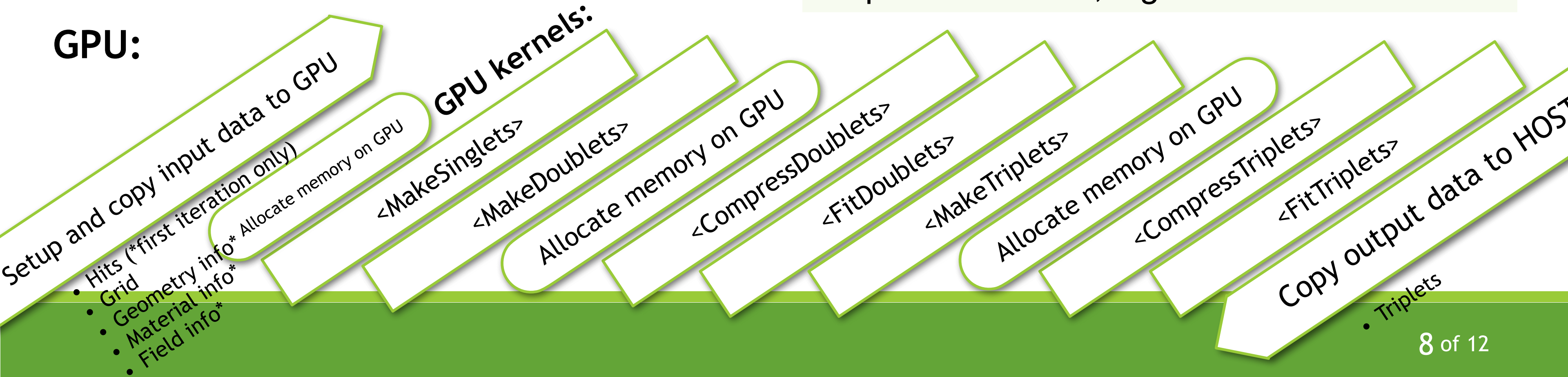
CPU implementation:

- Sequential calculation of the entire chain for each hit within a single thread: hit - singlet - doublets - triplets;
- Low efficiency of intra-event parallelization due to data access overhead.

GPU implementation:

- Each step of the chain as a single kernel;
- One element (hit, singlet, doublet, triplet) per GPU thread;
- Data arrays compression to get rid of empty elements (idle threads);
- Optimal for GPU, high overhead on CPU.

GPU:

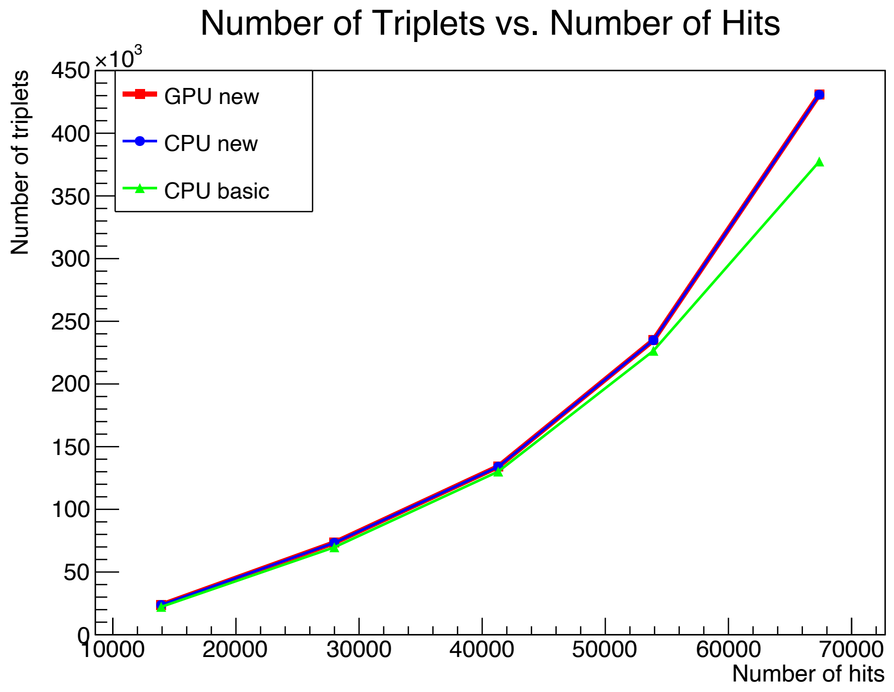


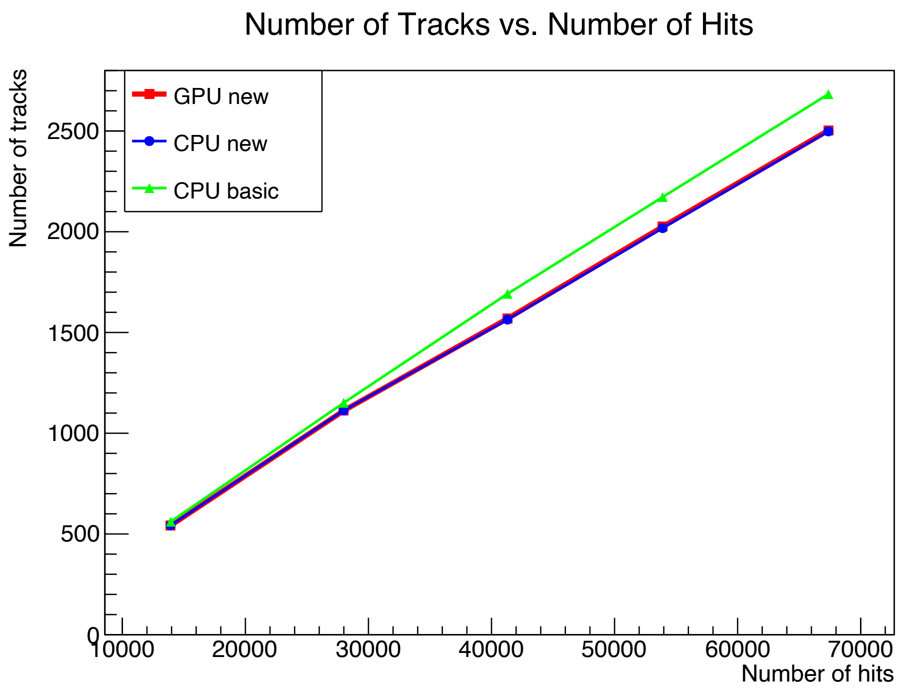
First tests and efficiency measurements

Testing:

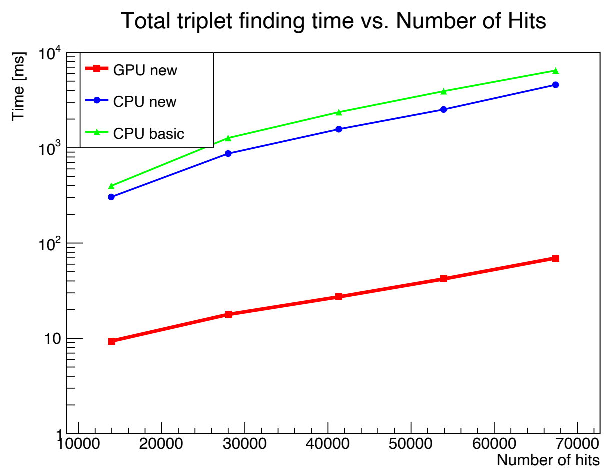
- 1 iteration;
- test data: 1 time slice (1 to 5 central events);
- comparison: new algo on GPU vs new algo on CPU (XPU OpenMP fallback) vs basic algo on CPU;
- CPU: 2x Intel Xeon Gold 6130; GPU: AMD Radeon VII

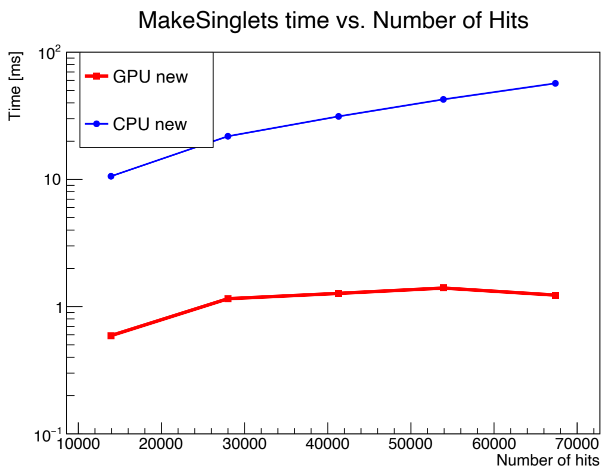
The algorithm is still under development, all measurements provided are not final results and serve only to evaluate intermediate status

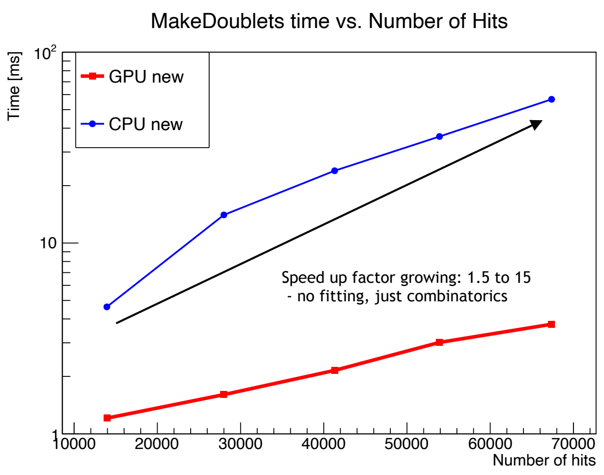


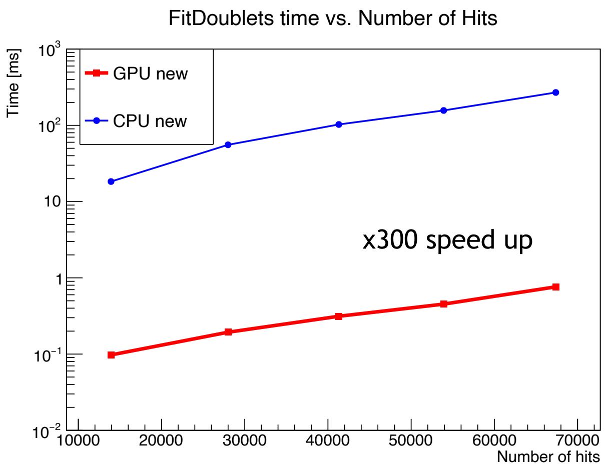


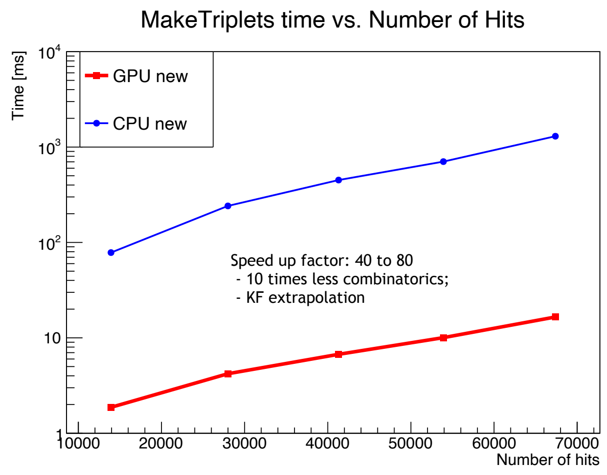
Runtime mesurements

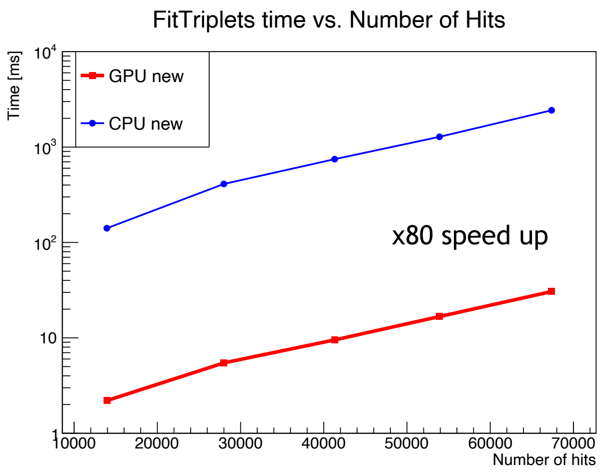












Energy consumption study (preliminary)

The study of the Kalman Filter track fitter's energy consumption is not only valuable on its own but also serves as a crucial step in developing and testing methodology for similar studies planned within the FIDIUM framework for the next year.

$$E = \boxed{\frac{k}{N}} \times \boxed{t \times \left(n_c \times P_c \times u_c + n_m \times P_m\right) \times PUE \times 0.001}^{1}$$

E — energy consumption (kWh)
t — task execution time (hours)
n_c — number of computational cores
P_c — power consumed by one computational core (W)
u_c — core utilization factor (from 0 to 1)
n_m — amount of memory used (GB)
P_m — power consumed by one gigabyte of memory (W)
PUE — Power Usage Effectiveness of the data center

k — number of tracks for energy consumption evaluation
N — total number of tracks processed per unit of time

$$C = E \times CI$$

C — total carbon footprint (gCO_2)
CI — carbon intensity factor

Goethe NHR:
- CPU: 2x Intel Xeon Gold 6148
- GPU: AMD MI210

PUE: 1.076
CI (Germany): 380 gCO_2/kWh

Test data:
CBM based
STS-like detector with 7 stations
Collision rate: 10 MHz
$N_{tracks/event}$: 1000
Memory usage:
- 10^6 tracks
- 240 byte per track
- ~230 MB

TDP:
CPU: 150 W / 20 cores = 7.5 W per core (preliminary)
GPU: measured with *rocm-smi*

[1] *Loïc Lannelongue, Jason Grealey, and Michael Inouye* **"Green Algorithms: Quantifying the Carbon Footprint of Computation"**

Conclusions

- Preparations for online data reconstruction, successful operations on GSI-VIRGO computing farm.
- Running event building and reconstruction tasks in Docker/Apptainer containers through Slurm, customization and building of containers by scripts.
- Translation CA tracking to GPU is in progress. The problem is technically solvable. The first results look very promising.
- Development of methodology and first studies of energy efficiency of reconstruction algorithms.

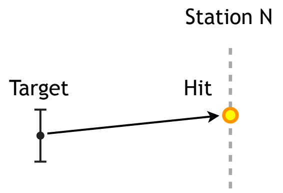
Next steps

- Continuing development of the GPU version of the CA track finder. Bug fixes, memory usage optimization, translation of other tracking stages to the GPU.
- Integration of GPU calculations into the online reconstruction chain, testing within of the next data challenges.
- Further work on containerization and automation of launching reconstruction tasks on computing farms, writing documentation.
- Further research and evaluation of energy efficiency of reconstruction algorithms.

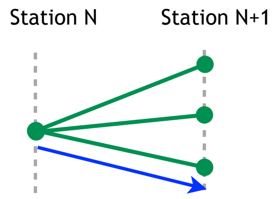
Backup

Triplets reconstruction on CPU

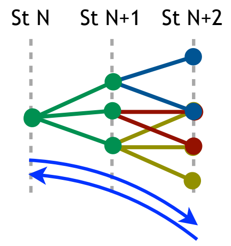
```
for loop by stations
    for loop by hits on station
        FindSinglets()
        FindDoublets()
        FitDoublets()
        FindTriplets()
        FitTriplets()
        StoreTriplets()
    End of for loop by hits on station
End of for loop by stations
```



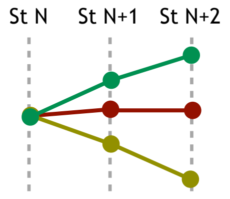
- Creating initial track parameters estimation. Fit a straight line through the target and the (left) hit taking into account magnetic field



- Search for possible combinations of hit pairs
- Fitting forward, adding a second (middle) hit to the parameter estimation



- Search for possible 3-hit segments
- Fitting forward and backward several times to get a final parameters estimation of the triplets



- Checking triplets, saving triplets whose parameters satisfy specified conditions

Triplets reconstruction on GPU (under development)

Setup and copy input data to GPU

- Hits (*first iteration only)
- Grid
- Geometry info *
- Material info *
- Field info *

GPU kernels:

Allocate memory on GPU

<MakeSinglets>

<MakeDoublets>

Allocate memory on GPU

<CompressDoublets>

<FitDoublets>

<MakeTriplets>

Allocate memory on GPU

<CompressTriplets>

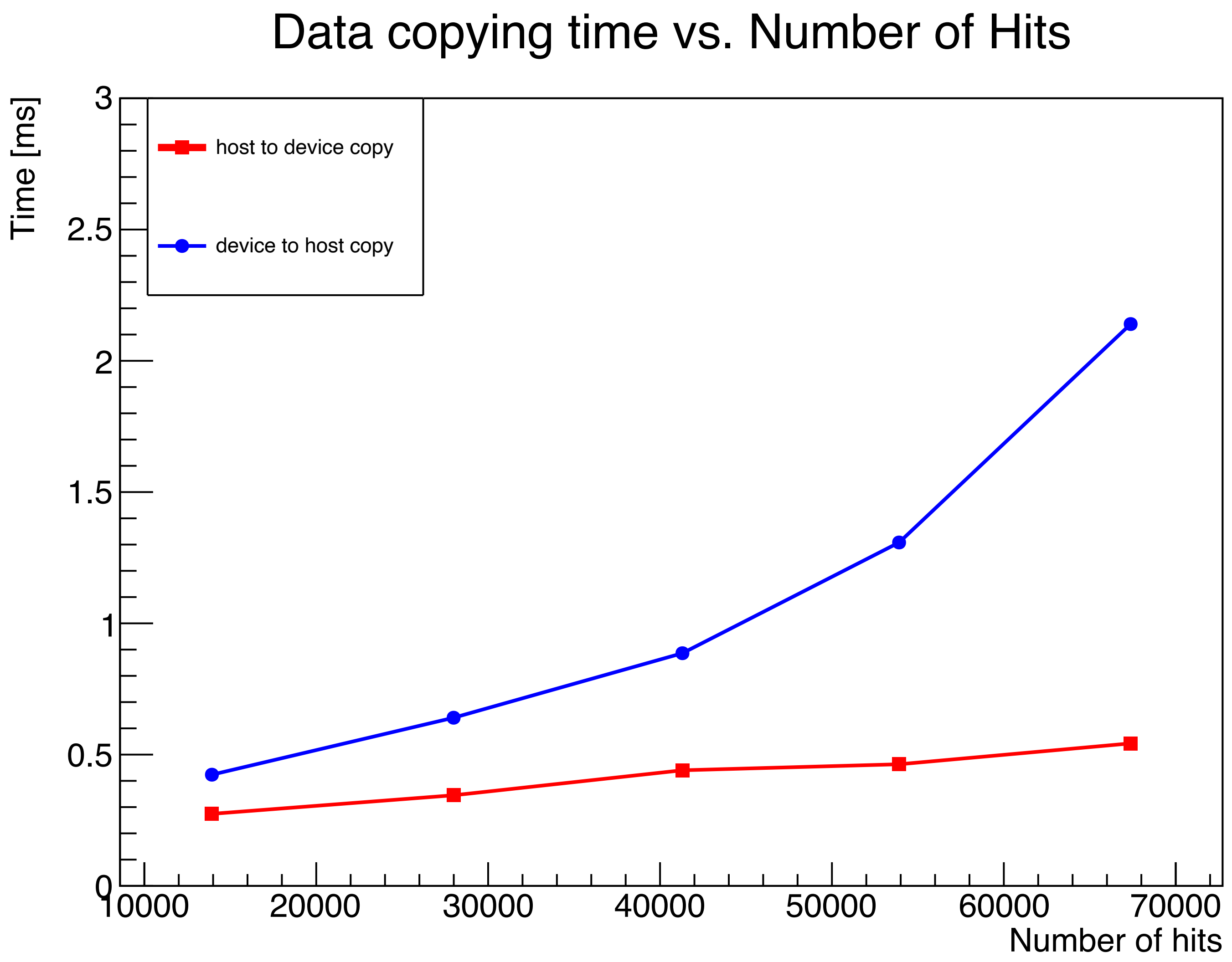
<FitTriplets>

Copy output data to HOST

- Triplets

- Only grid and iteration settings changes from iteration to iteration; other data could be copyed once
- Allocate memory on GPU for singlets and doublets; maxDoubletsPerSinglet = 150
- Creating singlets: processing one hit per GPU thread
- Creating doublets: processing one singlet per GPU thread
- Allocate memory on GPU for non empty doublets and triplets; maxTripletsPerDoublet = 15
- Get rid of “empty” doublets
- Fit doublets, add middle hit to parameters estimation
- Creating triplets: processing one doubler per GPU thread
- Allocate memory on GPU for non empty triplets and triplets; maxTripletsPerDoublet = 15
- Get rid of “empty” triplets
- Fit triplets forward and backward several times, check parameters; processing one triplet per GPU thread
- Copy results to HOST for furter track calculations

Runtime mesurements


Data copying time vs. Number of Hits
Time [ms]
Number of hits
host to device copy
device to host copy