The Patatrack Project

Andrea Bocci[1], Angela Czirkos[1], Antonio Di Pilato[4], Felice Pantaleo[1], Gabrielle Hugo[1], Marco Rovere[1], Martin Kwok[2], Matti Kortelainen[2], Vincenzo Innocente[1] , **Wahid Redjeb**[1,3]

[1]CERN, European Organization for Nuclear Research, Meyrin, Switzerland
[2]Fermilab, Fermi National Accelerator Laboratory, Batavia, IL, USA
[3]RWTH Aachen University, III. Physikalisches Institut A, Aachen, Germany
[4]CASUS, Center for Advanced Systems Understanding, Görlitz, Germany

On Behalf of the CMS collaboration

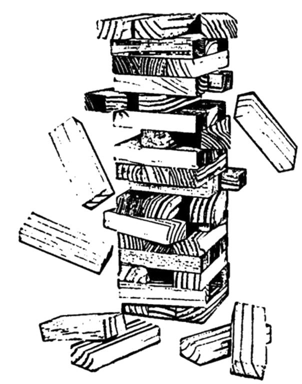
March, 21, 2022
CERN Openlab Technical Workshop

Patatrack

Patatrack is:

- A small group of passionate people
  - from Aachen, CERN, Fermilab, **CASUS**
    - Partnership with E4 and Intel
      - Software Support
      - Hardware testing
      - Student projects
  - Interested in **Heterogeneous Computing**, **Performance Portability** and **Software engineering**
- Ideas incubator
  - New algorithms
  - Software development for accelerators
  - Performance Portability

- **Alpaka developers!**
- **New CMS institute**



The Patatrack main goals are:

- Promote the use of accelerators in CMS
  - Hackathons, Knowledge Transfer
- Develop an online/offline **Heterogeneous Reconstruction**
  - Targeting Run 3 and Phase-2

10th Patatrack Hackathon!

Very successful hackathon in November 2021!

- Restricted to 10 people due to Covid
- Main topics:
    - Performance Portability with Alpaka
    - MPI
    - More developments on the Pixel-tracks reconstruction

10th Patatrack Hackathon

1 Nov 2021, 09:30 → 4 Nov 2021, 18:20 Europe/Zurich

3179/1-D06 (CERN)

Participants: Ali A A A M Marafi, Andrea Bocci, Angela Czirkos, Antonio Di Pilato, Christopher Rhys Sandever, Eric Cano, Felice Pantaleo, Marco Rovere, Wahid Redjeb

1. Welcome
Felice Pantaleo (CERN), Luca Malgeri (CERN), Silvio Donato (Universita & INFN Pis...
01/11/2021, 09:30

2. Hacking
01/11/2021, 09:40

3. Hacking
01/11/2021, 13:30

76. Scrum
01/11/2021, 17:00

5. Hacking
02/11/2021, 09:00

6. Hacking
02/11/2021, 13:30

77. Scrum
02/11/2021, 17:00

8. Hacking
03/11/2021, 09:00

11. Hacking
03/11/2021, 13:30

78. Scrum
03/11/2021, 17:00

10. Hacking
04/11/2021, 09:00

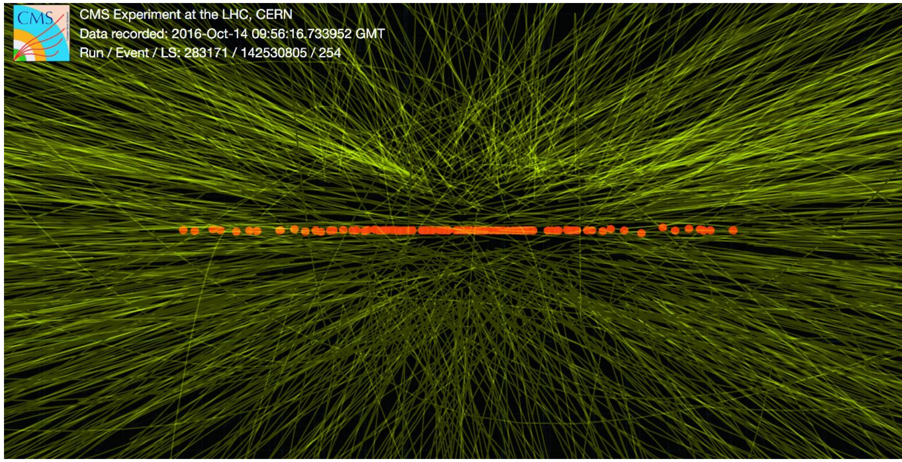
13. Hacking
04/11/2021, 13:30

24. Final Scrum and Conclusion
Felice Pantaleo (CERN)
04/11/2021, 16:30

CMS Pixel Tracks And Vertices Reconstruction

Testing the Alpaka Library

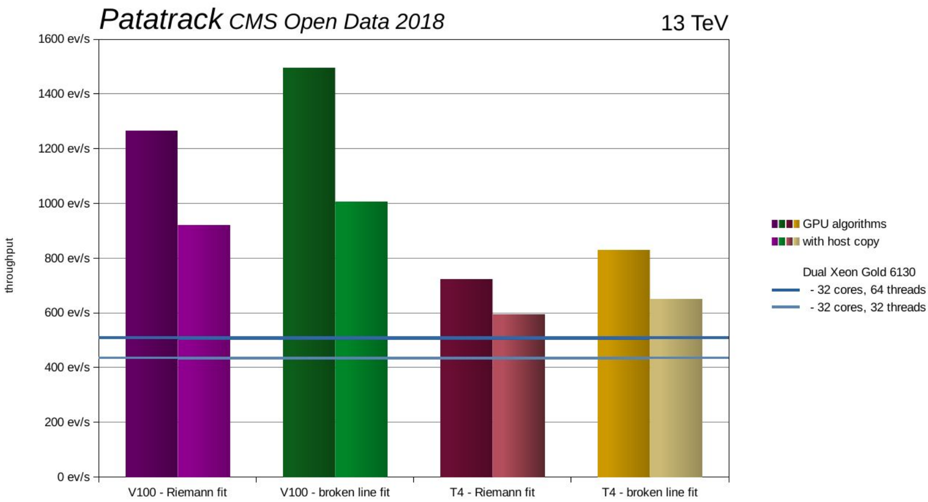
- Alpaka is a Performance Portability Library
  - Implement an abstraction layer
  - Obtain a single source code
    - Can be compiled for different architectures and backends
- Standalone version of the Patatrack Pixel Tracks and Vertices Reconstruction
  - CMS Software Module (Algorithms)
  - Light version of the CMS Software Framework
- Comparing performances wrt :
  - CUDA Native implementation (for NVIDIA GPUs)



Patatrack Pixel Tracks and Vertices Reconstruction

- Starting from energy deposits in the tracker
- Reconstruction is performed in parallel on GPUs
- **Charged Particles Trajectories and Vertices**
- Copy back to the CPU

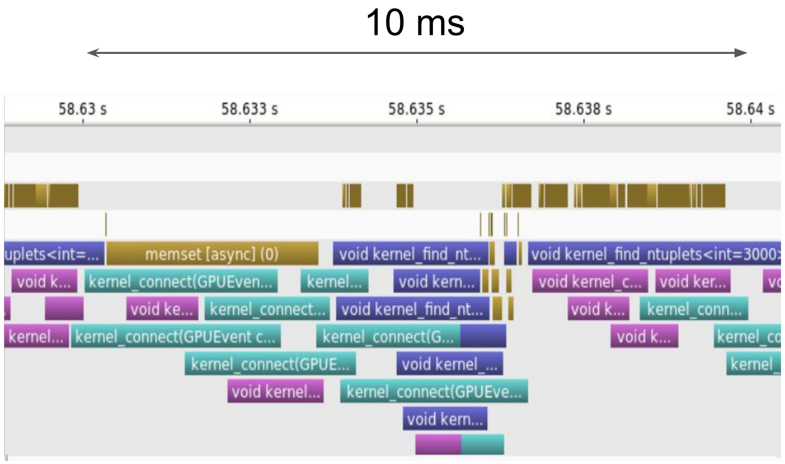
- Throughput up to 1.8x times higher wrt 2018 CMS HLT Pixel Reconstruction on dual socket Intel Xeon Gold 6130
- 2.9x higher on a Single NVIDIA Tesla T4!



CMS Pixel Tracks And Vertices Reconstruction Porting with Alpaka

Test Alpaka on the Patatrack Pixel Tracks and Vertices Reconstruction

- Not a simple embarrassingly parallel application
  - Long chains of parallel kernels with threads interacting with each other
  - Contains instruction that may/may not or may be different in alpaka
    - Atomics, barriers, …
- The code is completely different from the standard HPC
  - A lot of small kernels that run one after the other (streaming application)



Lot of parallel functions in only 10ms!

Allow us to test Alpaka on multiple aspects

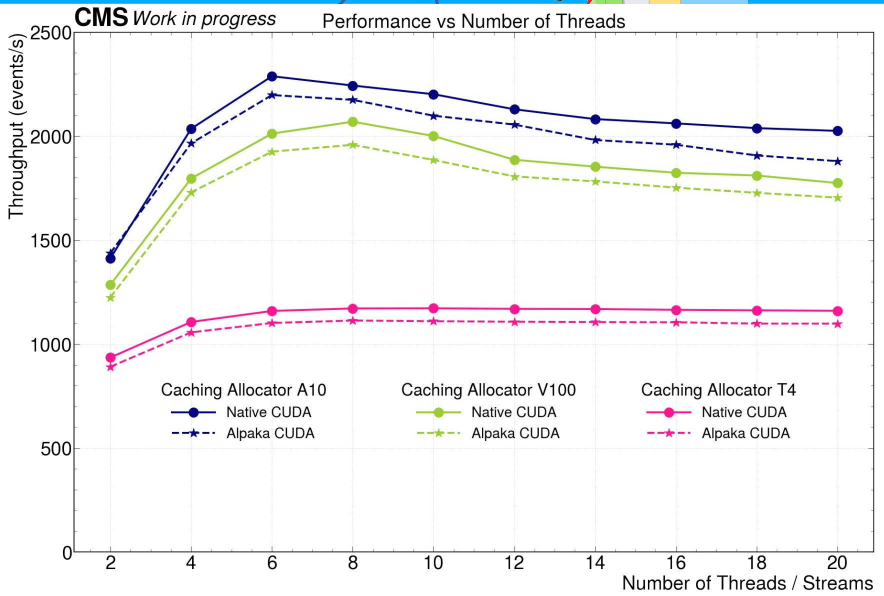
Goals

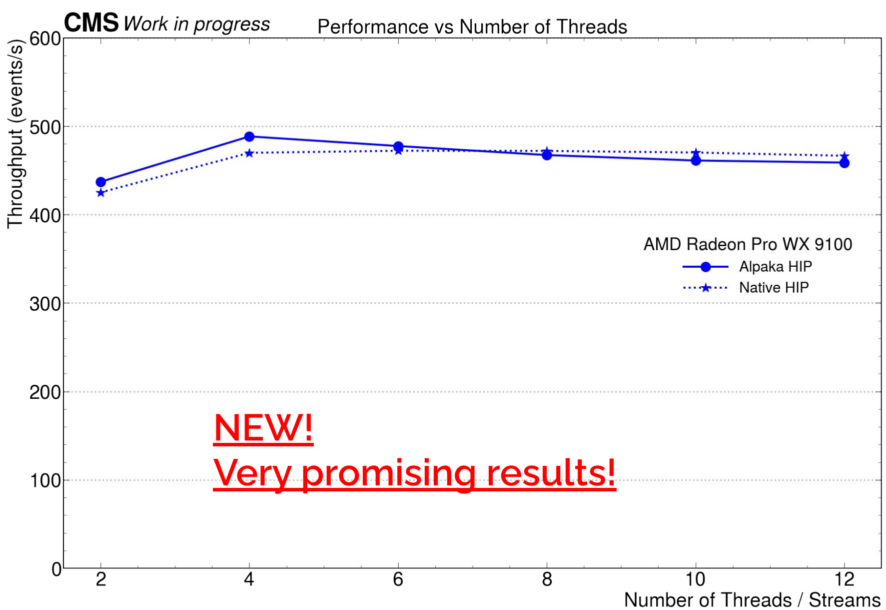
- Obtain a single source code for different accelerators
- Achieve compatible performance wrt the native ones
- Explore the best way to interface alpaka with the CMSSW Framework
- Gain expertise in performance portability for future developments and maintenance

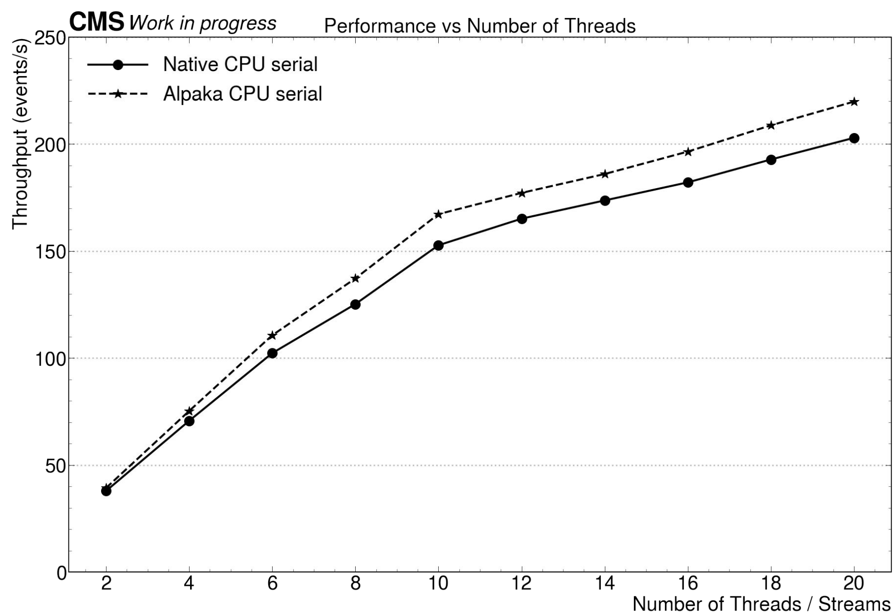
CMS Pixel Tracks And Vertices Reconstruction Porting with Alpaka - Results

Alpaka Pixel Tracks and Vertices Reconstruction

- **The interface with the CMS framework and the whole reconstruction have been ported to Alpaka!**
  - Single Source Code can run on:
    - NVIDIA GPUs (A10, V100, T4)
    - AMD GPU (Radeon Pro WX 9100)
    - multicore CPU (2x10 cores, 2x20 threads)
  - Same physics performance w.r.t native versions
  - Achieved **> 95% performance** wrt native implementation on all the GPUs tested
  - Alpaka CPU-serial version shows better performance w.r.t the native version







Future Plans

- Integration of Alpaka in CMSSW
- R&D on the pixeltrack-standalone demonstrator for Intel oneAPI
  - through the Alpaka interface via the SYCL backend
  - using native Intel oneAPI
- Achieve performance portability with Alpaka for new algorithms (already written in CUDA)
  - ECAL local reconstruction
  - HCAL local reconstruction

Conclusions

- CMS is evaluating solutions to develop a sustainable heterogeneous reconstruction
  - Sustainability, Maintainability and Testability are mandatory
  - Performance Portability libraries are a solution
  - Allow to run on a plethora of hardware architectures with a change in the configuration

- The reconstruction of pixel tracks and vertices has been ported to Alpaka:
  - Achieves close-to-native performance
  - Demonstrates the possibility of writing a maintainable single source code executable on different architectures
  - Using performance portability libraries for the Offline Reconstruction will allow us to exploit "supercomputers" and HPC data centers

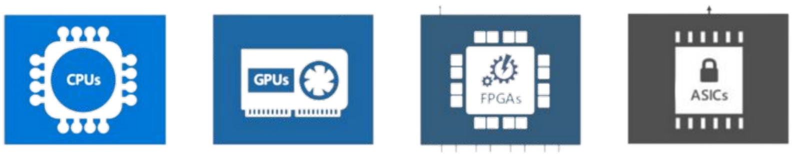
- Alpaka has been selected for performance portability in CMSSW for Run-3!

- Special thanks to Openlab and all the sponsors for support given to the Patatrack group!

BACKUP

Heterogeneous Computing in CMS

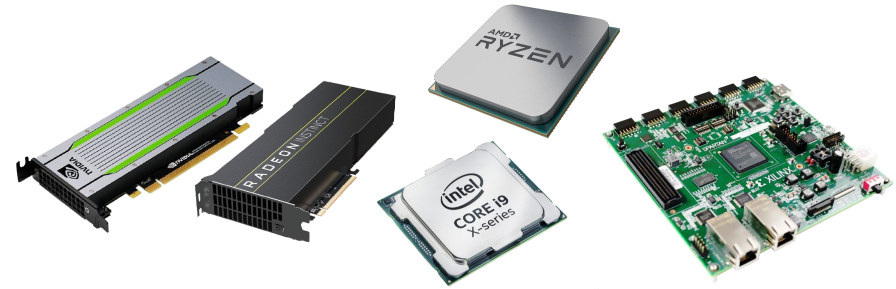
- Lot of R&D in the last years towards an **Heterogeneous Reconstruction**
  - New Algorithms and Dataformats
    - Pixel Tracks and Vertices Reconstruction (from RAW data) on GPU (Run 3)
    - HCAL and ECAL local reconstruction on GPU (Run 3)
    - HGCAL Local Reconstruction (Phase 2)
    - PF Clustering (Run 3)
  - Heterogeneous CMS Software Framework
    - Framework offloading to external device
    - Concurrency: keep CPU busy while running on GPUs



- Heterogeneous HLT Farm starting from Run 3
  - **To achieve better physics and computing performance at the same cost**
  - Design a reconstruction that can run at supercomputers and heterogeneous data centers

The issue of heterogeneous computing

- Several accelerators available on the market
  - CPUs, GPUs, FPGAs, ASICs
- And also several vendors
  - Intel, AMD, ARM, NVIDIA, ...
- And we must be ready for new devices too!



- Different drivers, binary formats, APIs, ...
- A lot of platforms available
  - CUDA, HIP, openCL, openGL, ...

- Writing different implementations for different architectures is not sustainable
  - Hard Maintainability, Testability and Validation
  - Code Duplication

<u>Write software once and use it on different architectures</u>

The solution:
Performance portability libraries

- **<u>Portable code is the solution</u>**
  - Long-Term maintainability, and testability
  - Avoid code duplication
  - Same algorithms for different hardware
  - Support for new devices

- **<u>Performance Portability Libraries</u>**
  - Abstraction layer to hide backend implementation
  - Map algorithm on different devices
    - Express parallelism on all the backends
  - Allow algorithm tuning for each device/backend

"Portability in high-level computer programming is the usability of the same software in different environments. The prerequirement for portability is the generalized abstraction between the application logic and system interfaces. When software with the same functionality is produced for several computing platforms, portability is the key issue for development cost reduction." ***Wikipedia***

- CMS offline and computing and HLT are **evaluating performance portability libraries for near term future**
- **Identified evaluation metrics:**
  - Programming Experience / Ease of development
  - Performance w.r.t Native implementation
  - Performance of fall back to CPU
  - CMSSW Integration
    - In framework and in building infrastructure
  - Supported Backends
    - CPU,GPU,FPGA and all the combination with vendors
  - User support
  - Long term prospects

Alpaka Performance Portability Library

- Developed by CASUS, Helmholtz-Zentrum Dresden – Rossendorf
- Header-only C++17 Library, Open Source
- Offers support to a multitude of backends
  - CUDA, HIP, openMP, TBB, std::thread, BOOST.Fiber
- Implement an Abstraction Layer
  - **Hierarchical Redundant Parallelism Model**
- Data-Agnostic Memory model
  - Pointer Based memory-model

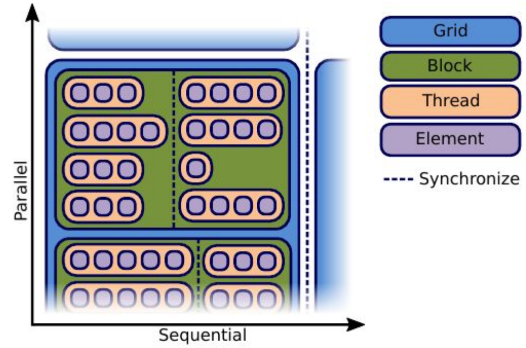
alpaka

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

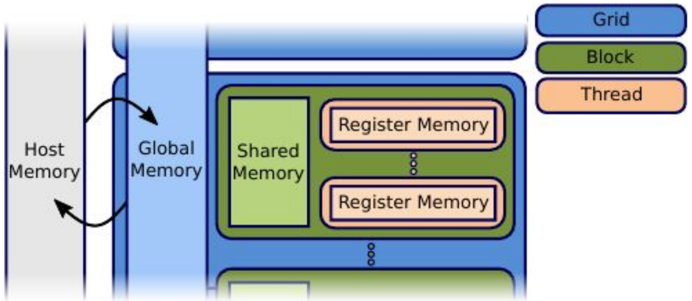
Alpaka Parallelization Hierarchy

- Grid of Blocks
- Each Block consists of Threads
- Each Thread processes multiple Elements
- Alpaka separates the parallelization strategy from the algorithm (*kernel*)



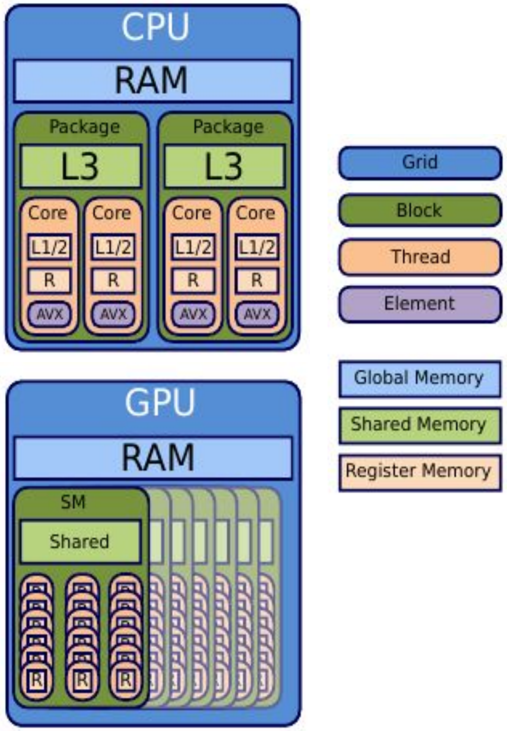
Alpaka Memory Hierarchy

- Global memory
- Shared Memory
- Register Memory



Mapping the Abstraction to the Hardware

- Alpaka separates the parallelization abstraction from the hardware capabilities
- Explicit mapping of the parallelization levels to the hardware
  - Set of predefined accelerators
  - The abstraction allows to support other accelerators for user-defined extension
  - Ignore specific unsupported levels of the model



Alpaka usage

- Initialization
  - Define the task Dimension (1D,2D,3D …)
  - Choose the host and device accelerators
  - Create the Queue (Stream)
    - And define its synchronization policy
- Writing Kernel (algorithm)
  - The kernel has to be a C++ functor
    - Templated on the accelerator
- Kernel Launch
  - Define Work Division (parallelization strategy)
  - Copy operations

```
using Dim=alpaka::DimInt<1u>;
using Idx=uint32_t;
using Size=std::size_t
using Acc=alpaka::AccGpuCudaRt<Dim,Size>;
using Host=alpaka::AccCpuSerial<Dim, Size>;
using Queue=alpaka::QueueCudaRtNonBlocking;
```

```
struct Kernel{
   template<typedef T_Acc>
   ALPAKA_FN_ACC void operator()(T_Acc const& acc,
        //arguments  ) const {
        //Your algorithm here
   }
};
```

```
alpaka::WorkDiv<Dim> workDiv(blocksPerGrid,
threadsPerBlockOrElementsPerThread,
elementsPerThread);
```

```
using BufHost = alpaka::Buf<DevHost, Data, Dim, Idx>;
BufHost bufHostA(alpaka::allocBuf<Data, Idx>(devHost, extent));
using BufAcc = alpaka::Buf<Acc, Data, Dim, Idx>;
BufAcc bufAccA(alpaka::allocBuf<Data, Idx>(devAcc, extent));
//copy from host to device
alpaka::memcpy(queue, bufAccA, bufHostA, extent);
```

```
auto const taskKernel =
alpaka::createTaskKernel<Acc>(
        workDiv,
        kernel,
        //arguments);
alpaka::enqueue(queue, taskKernel);
```

Testing the Alpaka Library

- Standalone version of the Patatrack Pixel Tracks and Vertices Reconstruction
  - CMS Software Module (Algorithms)
  - Light version of the CMS Software Framework
- Comparing performances wrt :
  - CUDA Native implementation (for NVIDIA GPUs)

Patatrack Pixel Tracks and Vertices Reconstruction from RAW data

1. Pixel RAW Data transferred to the GPU
2. RAW data decoding → DIGIs
3. Clustering of nearby Pixel digis → Clusters (Pixel hits)
4. Linking of doublets of Hits on different layers → Doublets
   a. *Fishbone-ntuplets*
5. Cellular Automaton pattern recognition for linking doublets segments → ntuplets
6. Fitting of found ntuplets → Pixel Tracks
   a. Multiple Scattering-Aware fit : *Broken Line Fit*
7. Clustering of Pixel Tracks → Pixel Vertices
8. Copy of the results back to the CPU
   a. On-demand conversion to legacy data format