Hardware-portable Data Analysis Building Blocks for the High Luminosity LHC

Mohammad Nasir Jan Momed

FH SciComp Workshop

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DASHH Data Science in Hamburg HELMHOLTZ Graduate School for the Structure of Matter

A few words about this PhD

- Computer science domain
- With the DASHH programm
- Starting stage
  - ~4months into this
- Today’s presentaion:
  - The first application
    - Physics domain

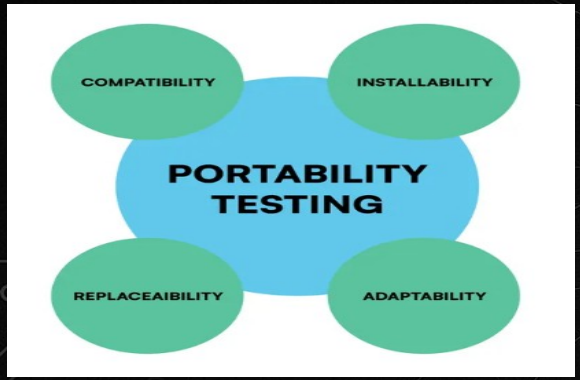
Outline

- Hardware Portability
- The First Building Block of this project
  - Phase II CMS Outer Tracker Unpacker
- Shifting the Unpacker to a portability ecosystem

Hardware Portability

Portability

- One code, various devices
- Write for one, run in many
- Very handy at long duration experiments like
  - LHC



Perfomance portability

- Not just run but run fast
- Keep code clean, get speed ups on CPU/GPU
- Finds optimal hardware-specific performance
- Abstraction layer programming

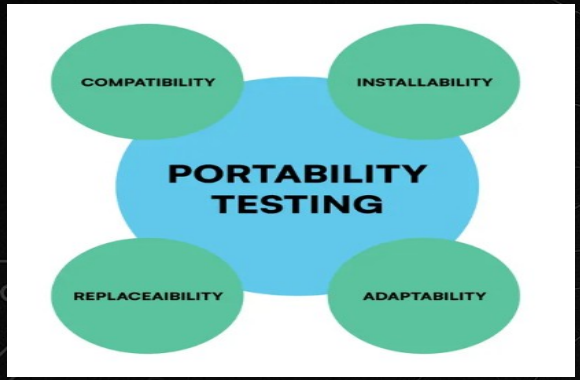
Portable ecosystems

- Abstraction Library for Parallel Kernel Acceleration (alpaka)
- KOKKOS
- SYCL
- OpenMP
- CUDA

Hardware Portability

Portability

- One code, various devices
- Write for one, run in many
- Very handy at long duration experiments like
  - LHC

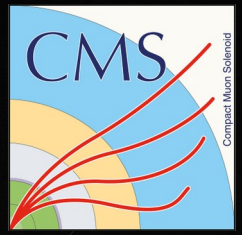


Perfomance portability

- Not just run but run fast
- Keep code clean, get speed ups on CPU/GPU
- Finds optimal hardware-specific performance
- Abstraction layer programming

Portable ecosystems

- Abstraction Library for Parallel Kernel Acceleration (alpaka)
- KOKKOS
- SYCL
- OpenMP
- CUDA



The First Building Block

Phase II CMS Outer Tracker Unpacker

Homogeneous compute engine environment → Heterogenous compute engine environment

CMS Upgrade for High Lumi LHC

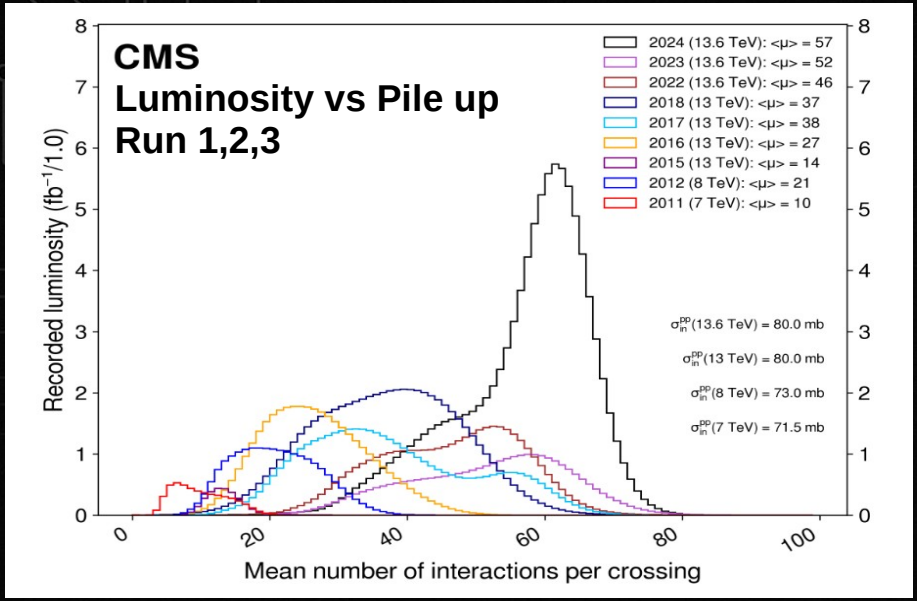
Why upgrade CMS

Increasing Pile up → Increasing Particle rate → Computational challenges

$$PileUp = \frac{interactions}{bunch\ crossing}$$

Run2 → Run3 → Run4 (HL-LHC)

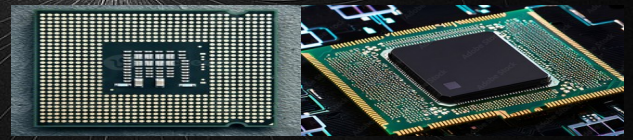
~30 → ~60 → Up to 200 (expected)



- Current tracker design
  - 20-30 Pile ups
- Higher Pile up requires:
- Higher detector granularity
  - To reduce occupancy
- Increased bandwidth
  - To accomodate higher data rates
- Needs
  - Optimized computing devices
  - Heteregenous architecture
  - Heteregenous programming

FPGA

CPU



GPU

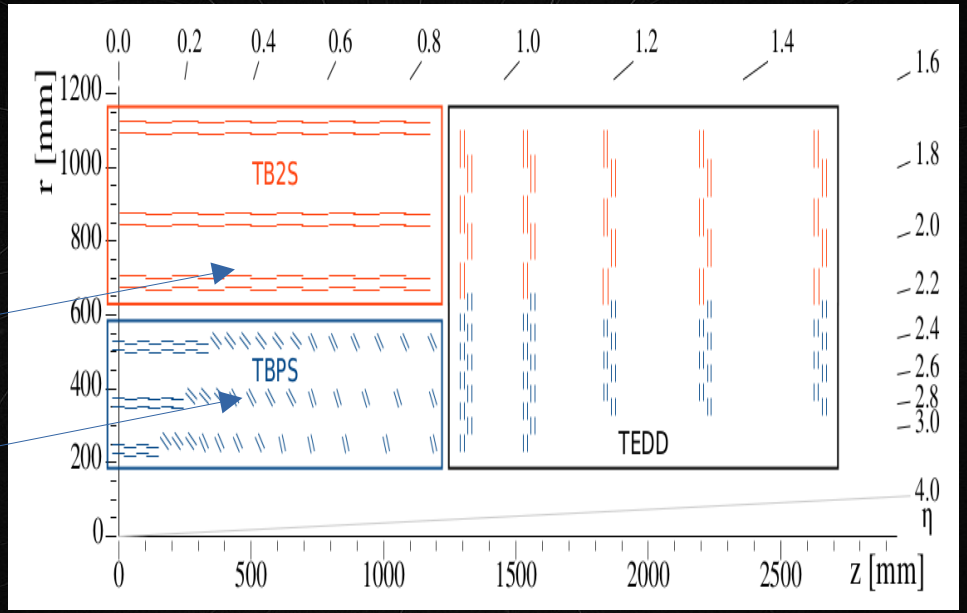
Tracking System for CMS High Lumi LHC

Upgrade :

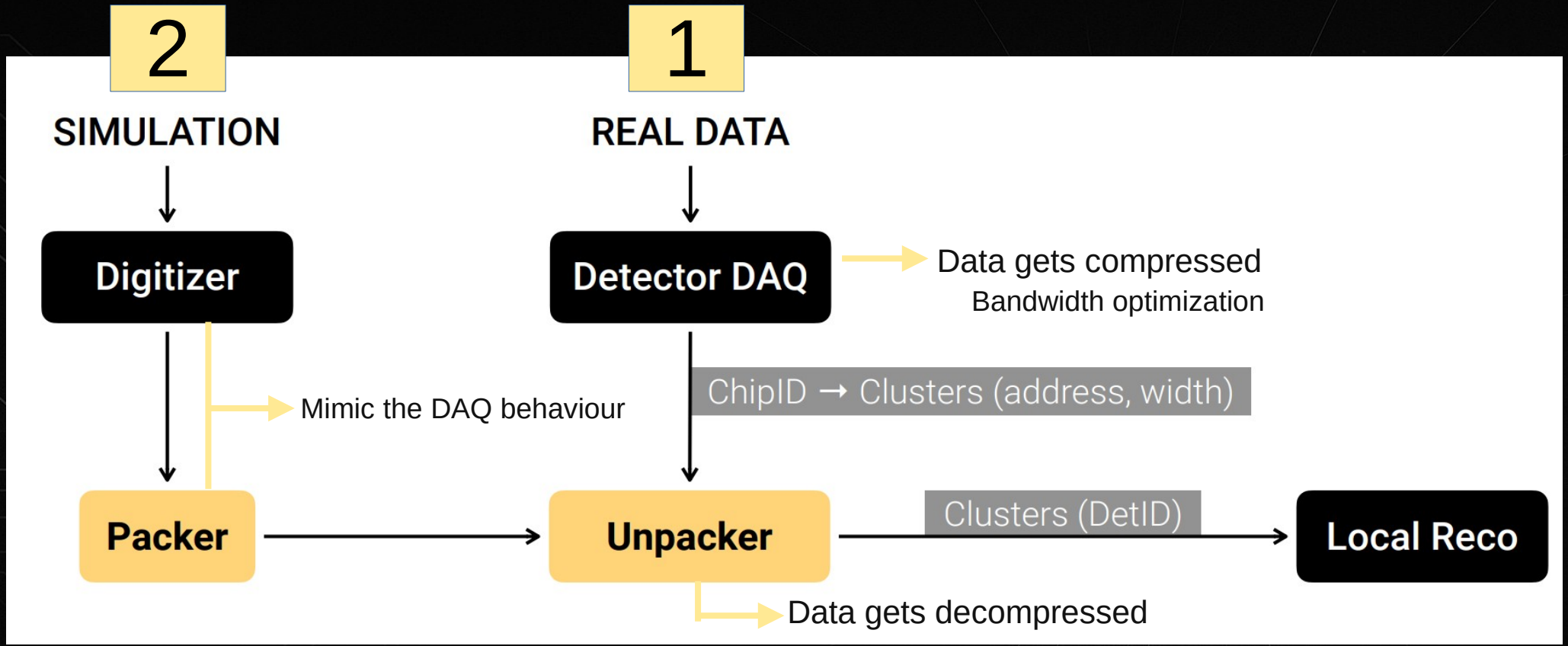
A completely new tracker is build

- Increased radiation hardness and forward acceptance
- Higher granularity
- Compatibility with higher data rates

- Inner Tracker (IT): silicon pixel modules
- Outer Tracker (OT): silicon modules with strip and macro-pixel sensors
  - 2 modules:
    - Two strips (2S) modules
    - Strip and Macro pixel sensor (PS) modules



Data Flow in CMS



Packer/Unpacker Role

Packer

- Packs the data in ClusterWords of 68 bits
  - Header part
  - Payload part

A Cluster Word in a 2S module

header												payload			
start sequence	error / status bit									L1 ID	nb clust		chip ID	Sclust address	width	
start séquence with 27 bits at '1'	0	CBC 8	CBC 7	CBC 6	CBC 5	CBC 4	CBC 3	CBC 2	CBC 1	CIC		0				.....
28 bits	9 bits									9 bits	7 bits	1 bit	3 bits	8 bits	3 bits	

Unpacker

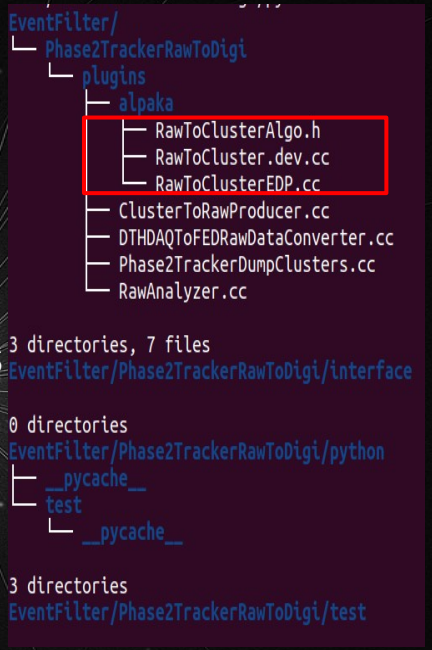
- Outputs physically relevant variables
  - Cluster width
  - Cluster position
- Converts raw data into usable detector clusters
- Prepares data for local reconstruction

CMSSW

CMSSW
- CMS framework for data:
  - Processing
  - Analysis
- Mainly C++ based but also python
- Used for:
  - Simulation
  - Reconstruction
  - Physics analysis

- Giant environment
  - ~2.5k Packages (top level directories)
  - ~ 5-7 million lines of code
- Devided in:
  - Packages
  - Subpackages
  - Files

  e.g. : Unpacker is just some files in the subpackage Phase2TrackerRawToDigi in package EventFilter.



alpaka

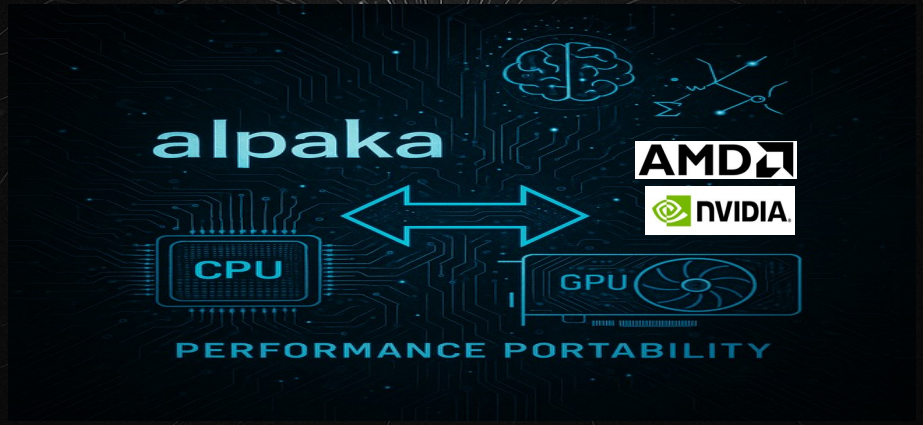
Abstraction Library for Parallel Kernel Acceleration

- Header-only C++20 abstraction library
- Offers Performance Portability
- Targets heteregenous architecture
- Explores all underlying level of parallelism
- Compile time flags can be used for different accelerators

  e.g.
  DALPAKA_ACC_GPU_CUDA_ENABLED

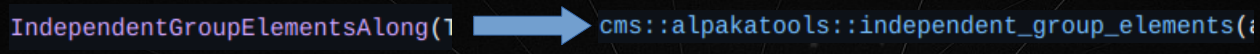
Key features

- Zero runtime overhead
  - abstraction resolved at compile time
- Flexible backend support: CUDA, OpenMP, HIP, etc
- Hierarchical parallelism: grid, block, thread, element levels
- Enables unified code for both CPU and GPU



CMSSW Adaption of Alpaka

- Used for the Heteregenous architecture in CMS
- A bit different from the raw alpaka



- Initial goal:
  - To migrate the GPU codes from CUDA, to be used in
    - Nvidia GPUs
    - AMD GPUs
- Selected over other Portability libraries as a result of CMS internal evaluation process

Shifting the Unpacker to a Portablity ecosystem

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DASHH Data Science in Hamburg
HELMHOLTZ Graduate School for the Structure of Matter

The Present Homogenous Code (Unpacker)

RawToCluster.cc

- Unpacker file
  - In sub-package Phase2TrackerRawToDigi
    - In package EventFilter

```
EventFilter/
└── Phase2TrackerRawToDigi
    └── plugins
        ├── ClusterToRawProducer.cc
        ├── DTHDAQToFEDRawDataConverter.cc
        ├── Phase2TrackerDumpClusters.cc
        ├── RawAnalyzer.cc
        └── RawToClusterProducer.cc

2 directories, 5 files
EventFilter/Phase2TrackerRawToDigi/interface

0 directories
EventFilter/Phase2TrackerRawToDigi/python
├── __pycache__
└── test
    └── __pycache__

3 directories
EventFilter/Phase2TrackerRawToDigi/test
```

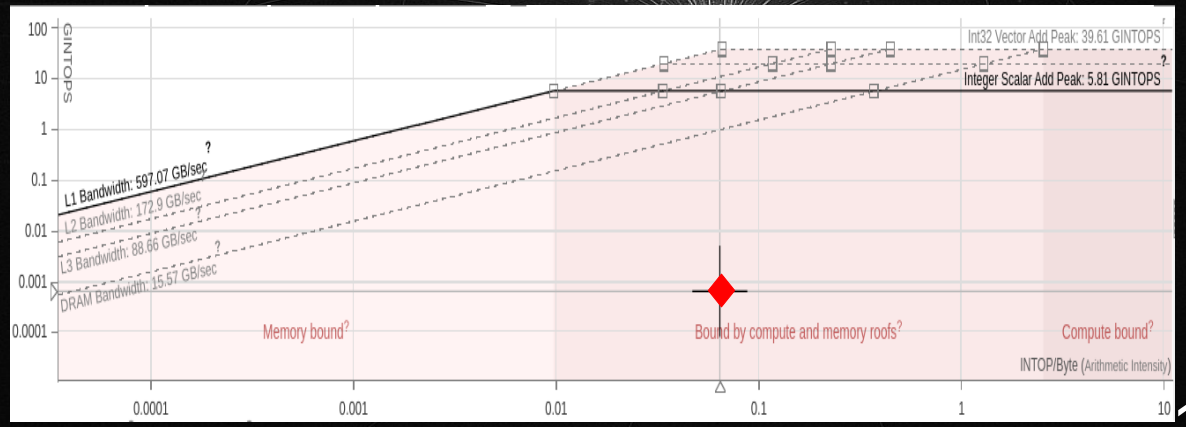
Motivation to port

- More heteregenous architecture in HL-LHC
- Performance can get better with alpaka

Roofline model

- ➔ perfomance as a function of artimetic intensity
- ➔ Integer operations per seconds vs Integer operations per byte

- It shows, the code is sub-optimal
- Heteregenous architecture
  - Reduce the bottlenecks
  - Higher memory bandwidth
  - Higher compute throughput



Method Used for Porting

CPU

Original Code

- Once for the process
  - Build a stack map of Detector Ids
- Per each Event
  - Loop Over DTCs (216 ≥)
  - Loop Over Slinks (4 ≥)
  - Read Header & Offset words
  - Loop over channels (36 ≥)
  - Get DetIDs
  - Read channel Header
  - Read Strip Payloads
  - Read Pixel Payloads
  - Unpack Per Module (2S/PS)

CPU

Ported Code

- Once for the process
  - Build a stack map of Detector Ids
  - Loop Over DTCs (216 ≥)
  - Loop Over Slinks (4 ≥)
  - Loop over channels (36 ≥)
    - Get the module type/Info
    - Store it in a buffer for later use
- Per each Event
  - Get the RAW data
  - Store in a buffer (LinearData)
    - .data
    - .size
    - Offset

GPU

Parallelization per each Step

- Loop Over LinerData
  - Read Header & Offset words
  - Loop over channels (36 ≥)
  - Get DetIDs
  - Read channel Header
  - Read Strip Payloads
  - Read Pixel Payloads
  - Unpack Per Module (2S/PS)

Current Progress

Original Code

CPU

- Once for the process
  - Build a stack map of Detector IDs
- Per each Event
  - Loop Over DTCs (216 ≥)
  - Loop Over Slinks (4 ≥)
  - Read Header & Offset words
  - Loop over channels (36 ≥)
  - Get DetIDs
  - Read channel Header
  - Read Strip Payloads
  - Read Pixel Payloads
  - Unpack Per Module (2S/PS)

Ported Code

CPU

- Once for the process
  - Build a stack map of Detector Ids
  - Loop Over DTCs (216 ≥)
  - Loop Over Slinks (4 ≥)
  - Loop over channels (36 ≥)
    - Get the module type/Info
    - Store it in a buffer for later use
  - 216 X 4 = 864 parallel threads
- Per each Event
  - Get the RAW data
  - Store in a buffer (LinearData)
    - .data
    - .size
    - Offset

Parallelization per each step

GPU

- Loop Over LinerData
  - Read Header & Offset words
  - Loop over channels (36 ≥)
  - Get DetIDs
  - Read channel Header
  - Read Strip Payloads
  - Read Pixel Payloads
  - Unpack Per Module (2S/PS)

Successfully Ported

Under Debugging

Under Development

Evaluation

- Match head to head the current resutls with the CPU code
  - A current stage head to head matching is not clearly possible
  - What one can do is to match some variables using only the serial backend of alpaka
    To verify the correctness at this stage
- And with that at the current stage we get a complete match with reading the data

CPU

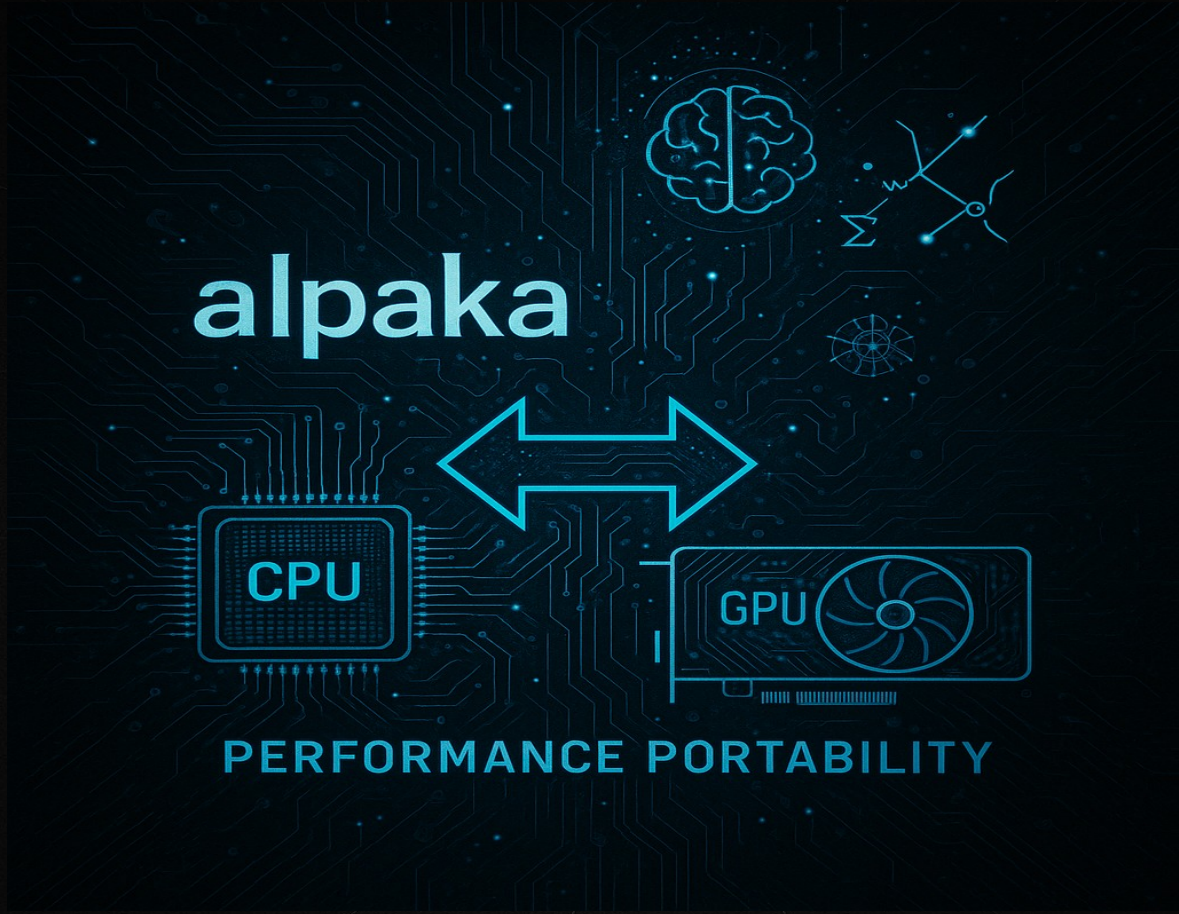
```
headerWords[0] value is: 4294967295
offsetWords[0] value is: 1310720
channelOffset16 is: 0
Idx is: 88
 numStripClusters = 21
 numPixelClusters = 17
channelOffset16 is: 20
Idx is: 168
 numStripClusters = 20
 numPixelClusters = 27
channelOffset16 is: 45
Idx is: 268
 numStripClusters = 22
 numPixelClusters = 22
```

Alpaka Serial

```
%MSG-i AlpakaService:  (NoModuleName) 24-Jun-2025
AlpakaServiceSerialSync succesfully initialised.
Found 1 device:
  - AMD EPYC-Milan Processor
%MSG
24-Jun-2025 15:55:43 CEST  Initiating request to
alistic_v1_STD_2026D98_PU200_RV229-v1/2580000/0b2
24-Jun-2025 15:56:25 CEST  Successfully opened fi
_v1_STD_2026D98_PU200_RV229-v1/2580000/0b2b0b0b-f
Begin processing the 1st record. Run 1, Event 950
headerWords[0] = 4294967295
offsetWords[0] = 1310720
ChannelOffset16 is: 0
idx is: 88
n strip clusters are: 21
n pixel clusters are: 17
ChannelOffset16 is: 20
idx is: 168
n strip clusters are: 20
n pixel clusters are: 27
ChannelOffset16 is: 45
idx is: 268
n strip clusters are: 22
n pixel clusters are: 22
```

Further Progress of the first building block

- Complete the porting
- Cross check the correctness
- Evaluate the performace
- Possible Optimizations




Thank You
alpaka
CPU
Port to GPU
CPU
CPU
CPU
GPU