Portable HCAL reconstruction in the CMS detector using the Alpaka library

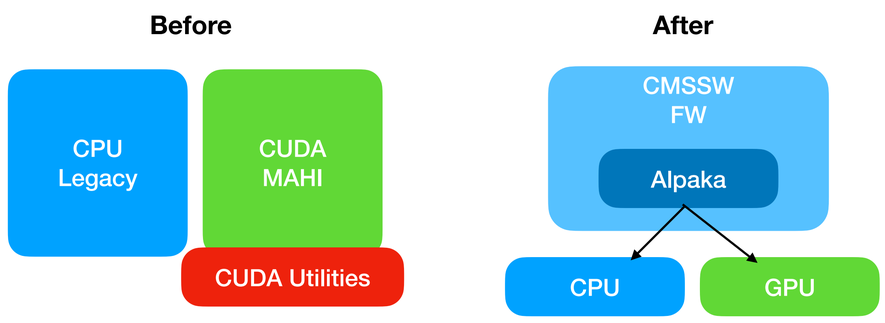
CHEP 2024
19 - 25 October, 2024 I Krakow, Poland

Martin Kwok (Fermilab)
on behalf of CMS collaboration

Introduction

Alpaka is a performance portability tool adopted by CMS for Run 3
- One implementation for *both* CPU and different GPU architectures

Goal of the project
- Re-engineer the CUDA implementation of HCAL local reconstruction using Alpaka
  - No algorithmic changes introduced, which simplifies the validation process
- Replace custom-CUDA utilities (data-migration, data-structure creation) with CMSSW tools



Implementation

There are 4 major components involved in the reconstruction
- DigiConverter: convert input data into Structure-Of-Array (SoA) format
- Calibration Data: produce HCAL conditions in SoA format
- Compute Kernel: compute HCAL energy reconstruction
- SoA to Legacy conversion: convert to legacy CPU format

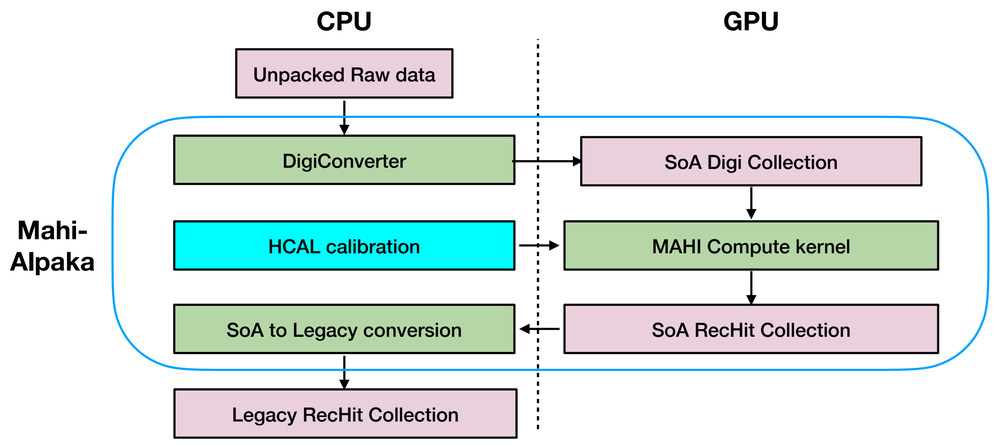
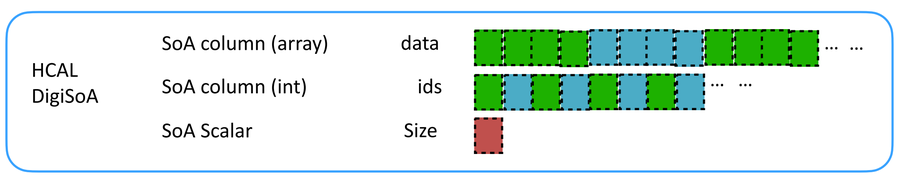


Illustration of steps involved in CMS HCAL local reconstruction.
Green boxes are data-product producers; red boxes indicate data product;
Cyan box is the auxiliary input data that depends on detector configuration/calibrations

DigiConverter

- Structure-Of-Array (SoA) is a more GPU-friendly data structure
- CMSSW has introduced generic mechanism to generate common SoA data-structure using macros
- We defined an HCAL DigiSoA to store in a single structure:
  - Per-channel numbers (e.g. channel id)
  - Per-channel arrays (e.g. energy in each bunch crossings)
  - Constant scalar (e.g. total channels)



Calibration data

- HCAL local reconstructions requires a lot of condition products
  - e.g. pedestal and response corrections
- CUDA implementation used simple arrays for *each* products
- Simplified from 18 event products into 4 SoA with multiple columns
- Simplifies access to related-SoA columns with custom class of SoA

HcalRecoParamWithPulseShapeSoA

Per-Channel parameters

Per *unique* PulseShape parameters

```
GENERATE_SOA_LAYOUT(HcalRecoParamSoALayout,
                    SOA_COLUMN(uint32_t, param1),
                    SOA_COLUMN(uint32_t, param2),
                    SOA_COLUMN(uint32_t, ids)
                    )
GENERATE_SOA_LAYOUT(HcalPulseShapeSoALayout,
                    SOA_COLUMN(HcalPSfunctorArray, acc25nsVec),
                    SOA_COLUMN(HcalPSfunctorArray, diff25nsItvlVec),
                    SOA_COLUMN(HcalPSfunctorBXarray, accVarLenIdxMinusOneVec),
                    SOA_COLUMN(HcalPSfunctorBXarray, diffVarItvlIdxMinusOneVec),
                    SOA_COLUMN(HcalPSfunctorBXarray, accVarLenIdxZEROVec),
                    SOA_COLUMN(HcalPSfunctorBXarray, diffVarItvlIdxZEROVec)
                    )
```

256 floats arrays

25 floats arrays

Compute Kernels

- Many infrastructural improvements with introducing algorithmic changes
  - Code structure
  - Data access methods (Simple index with SoA)
  - Wrapper around CUDA type-casting intrinsic functions for CPU/GPU backends
  - Apply Alpaka syntax to CUDA kernels
    - Kernel helper functions for simple work division
- Enhanced readability of kernels
- Use 500 less lines in the Alpaka implementation

CUDA

Computing data stride

```
// conditions based on the hash
// FIXME: remove hardcoded values
auto const qieType = qieTypes[hashedId] > 0 ? 1 : 0;  // 2 types at this point
auto const* qieOffsets = qieCoderOffsets + hashedId * HcalQIECodersGPU::numValuesPerChannel;
auto const* qieSlopes = qieCoderSlopes + hashedId * HcalQIECodersGPU::numValuesPerChannel;
```

Alpaka

Clean, simple index access

```
// conditions based on the hash
auto const qieType = mahi.qieTypes_values()[hashedId] > 0 ? 1 : 0;  // 2 types at this point
auto const* qieOffsets = mahi.qieCoders_offsets()[hashedId].data();
auto const* qieSlopes = mahi.qieCoders_slopes()[hashedId].data();
```

Validation

- Validated at the 2 different levels:
- Module level validation with O(1000) events
  - Compares per-channel rechit energy
  - Observed only numerical differences
- Trigger menu level validation with O(100k) events
  - HCAL local reconstruction is used by any triggers with jets
    Hence, a more stringent test
  - Compare changes of HLT decisions of every paths in the latest HLT menu
  - +/-2 events for most paths
  - Consistent with numerical differences
- Multiple comparisons done with each level:
  - Alpaka GPU v.s. CUDA
  - Alpaka CPU v.s. Legacy CPU
  - Alpaka GPU v.s. Alpaka CPU

Performance

- Measured *event throughput* of full HLT menu on dedicated computing node with same hardware (CPU and GPU) to the data-taking HLT farm
- The measurement node is fully loaded, similar to HLT runs in production
- Measured with and without NVIDIA Multi-Process Service (MPS) enabled
- Alpaka implementation brings ~2% speed-up in the **full** HLT menu
  - Only infrastructural improvements
  - Without algorithmic changes

	CPU	GPU without MPS	GPU with MPS	MPS speed up	GPU speed up w.r.t. CPU
HCAL legacy	388.8 ev/s	565.6 ev/s	635.6 ev/s	+12.4%	+63.5%
HCAL alpaka	387.5 ev/s	619.3 ev/s	649.3 ev/s	+4.8%	+67.6%
Alpaka speed up	-	+ 9.5%	+2.2%		

Conclusion

- Re-engineered CUDA implementation of HCAL local reconstruction to use Alpaka
- Single implementation for CPU and GPU(s)
- Major code simplifications taking advantage of new CMSSW FW capabilities
  - Common infrastructure for SoA, kernel helper functions
- Validated to reproduce CUDA/Legacy CPU code for 2024 data
- Deployed for data taking since August 2024
- Alpaka implementation brings ~2% speed-up in the full HLT menu

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Contact: kkwok@fnal.gov OR

FERMILAB-POSTER-24-0304-PPD