CLUE4HEP

Erica Brondolin, Felice Pantaleo, Marco Rovere

Key4hep Discussion, 22nd June 2021
Reconstruction in CMS HGCAL

- Particle shower reconstruction in **high-granularity calorimeter** is very interesting and crucial task in high-density environments
  - Typical situation at HL-LHC → Many showers tend to **overlap**
  - Standard reconstruction algorithms using combinatorics are expected to fail due to **memory/timing explosion**
  - **Fertile ground** for new techniques and algorithms: clustering, machine learning, graph theory, and modern computer architectures
    - Planned and designed, taking into account the information from the tracking system and timing detectors
- New flexible framework can be re-used in other (future) experiments using high-granularity calorimeters
What is TICL?

- TICL (The Iterative Clustering) is a modular framework integrated and under development in CMS software (CMSSW)
- Main purpose: processing calo 5D rechits \((x, y, z, t, E)\) and returning particle properties and probabilities
- In a nutshell: grouping 2D Layer Clusters into 3D clusters (Tracksters) iteratively to reconstruct different particle species

[Electron with \(E=32\) GeV; Test beam 2016 @FNAL]
**TICL modules/components**

**Flexible, efficient, versatile**

- Algorithms are designed as swappable plugins, with heterogeneous architectures / portability in mind
- Explore new algorithms or techniques (e.g. Machine Learning) with plug-in on top
- Skip and/or change modules easily
- Mostly geometry independent

**Diagram**

- Tracks: from Tracker
- PID
- Denoising
- Rechits
- 2D Clusters
- Seeding Region
- Tracksters
- Energy Regression
- Trackster Splitting
- MIP Tracksters

---

Erica Brondolin (erica.brondolin@cern.ch)
TICL modules/components

Example: CLUE

- Tracks from Tracker
- Seeding Region
- PID
- Denoising
- PFTICLProducer
- Timing from MTD/ETL
- Energy Regression
- Trackster Splitting
- MIP Tracksters

Flexible, efficient, versatile

- Algorithms are designed as swappable plugins, with heterogeneous architectures / portability in mind
- Skip and/or change modules easily

- Explore new algorithms or techniques (e.g. Machine Learning) with plug-in on top
- Mostly geometry independent
2D Clusters with CLUE

- CLUE (CLUstering by Energy) is an algorithm inspired by “Clustering by fast search and find of density peaks” (Ref.)

- Main characteristic:
  - Energy density - rather than individual cell energy - used to define ranking, seeding threshold, etc…

- GPU-friendly, i.e. suitable for the upcoming era of heterogeneous computing in HEP
Cluster Algorithm: CLUE

- Querying neighborhood is a frequent operation in density-based clustering → **fast**!
- Build **Fixed-Grid Spatial Index** for hits on each layer ($\eta, \phi$ space)
  - Grid tiles are small compared to the size of HGCAL layer
  - Each tile in the grid hosts indices of hits inside it and has a fixed length of memory to store the hosted indices
- To find the neighborhood hits $N_d(i)$ of $i$-hit, we only need to loop over hits in $\Omega_d(i)$
Cluster Algorithm: CLUE

- Calculate local energy density ($\rho$) in a distance ($d_c$)
  - Each hit $j$ weighted by the deposited energy ($E_j$)
  - For each hit, calculate $\rho_i$
  - Individual $d_c$ values considered in HGCAL Silicon and Scintillator sections

\[
\rho_i = \sum_{j \in N_d(i)} E_j \times f(d_{ij}) \quad ; \quad f(d_{ij}) = \begin{cases} 
1, & \text{if } i = j \\
k, & \text{if } 0 < d_{ij} \leq d_c \\
0, & \text{if } d_{ij} > d_c
\end{cases}
\]

$\rho_c$, $d_c$, $\delta_c$, $\delta_0$ are tunable parameters chosen with purity vs fake studies

(build data structure) density nearest higher find seed assign clusters
Cluster Algorithm: CLUE

- Calculate “Nearest-Higher” hit within $N_{dm}(i)$
  - Define $d_m = \max(\delta_o, \delta_c)$, $\delta_o$ and $\delta_c$ parameters for outlier demotion and seed promotion
  - Find the closest hit with higher local energy density, $n_{h_i}$
  - Calculate the separation distance $\delta_i = \text{dist}(i, n_{h_i})$

$\rho_c$, $d_c$, $\delta_c$, $\delta_0$ are tunable parameters chosen with purity vs fake studies
Cluster Algorithm: CLUE

- Promote as seed if $\rho_i > \rho_c$, $\delta_i > \delta_c$
- Demote as outlier if $\rho_i < \rho_c$, $\delta_i > \delta_o$
- Assign unique, progressive cluster ID to each cluster
  - Followers are defined and associated to their closest seed

$\rho_c$, $\delta_c$, $\delta_o$ are tunable parameters chosen with purity vs fake studies

→ Rock solid against noise
→ Clustering almost all energy

Example of decision plot
Stand-alone CLUE

- Repository: [https://gitlab.cern.ch/kalos/clue](https://gitlab.cern.ch/kalos/clue)
  - Tested successfully in 2020 with test beam data
- The pre-requisite dependencies are `$gcc7$, $gcc8.3$, $cuda10$, Boost, TBB
  - + CUPLA (C++ User interface for the Platform independent Library Alpaka)
- Compile with MakeFile to produce three executables from same source file (`main.cc`)
  - `main`: (nvcc) this will build the native C++ implementation of CLUE and its corresponding native CUDA one.
  - `mainCuplaCPUTBB`: (g++) this will build the native C++ implementation of CLUE and its corresponding TBB one built using CUPLA.
  - `mainCuplaCUDA`: (nvcc) this will build the native C++ implementation of CLUE and its corresponding CUDA one built using CUPLA.
- `useGPU`: external flag given at run time to select the ‘parallelise version’ of the code or not
Explore CLUE integration in key4hep

- Fork of standalone in [https://gitlab.cern.ch/ebrondol/clue](https://gitlab.cern.ch/ebrondol/clue)

- Plan:
  1. Compile CLUE in cmake
  2. Source key4hep and introduce EDM4HEP & PODIO as external libraries
  3. Create simulated EDM4HEP events
     - a. Very simple events
     - b. More realistic events
  4. Conversion code EDM4HEP <-> CSV
Compile CLUE in cmake

- Not straightforward to compile with nvcc and g++ the same target (`main.cc`)
- Solved with making a temporal copy of the main.cc file

```
# A copy with extension cu is needed to set the same source file in
# both languages (C++/CUDA)
configure_file(main.cc main.cu)
```


- More ingenious ideas are welcome!

- !!! Automatically use CUDA/nvcc only if found
Source key4hep and introduce EDM4HEP & PODIO as external libraries

- Code in `debug_podio` branch w/ EDM4HEP flag
- Changed to source only

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

- Open problem
  
  If only key4hep setup is sourced, `mainCuplaCPUTBB` does not run on parallelise option

```
source /cvmfs/sft.cern.ch/lcg/views/LCG_96/x86_64-centos7-gcc8-opt/setup.sh
```

is also needed. still investigating why
Source key4hep and introduce EDM4HEP & PODIO as external libraries

- Code in `debug_podio` branch w/ `EDM4HEP` flag
- Insert `podio::podioRootIO` in library for each executable gives the following error for CUDA

```
/afs/cern.ch/user/e/ebrondol/work/key4hep/20210517/clue/include/read_events.h(260): error: function "podio::EventStore::EventStore(const podio::EventStore &)"
/cvmfs/sw.hsf.org/spackages/linux-centos7-x86_64/gcc-8.3.0/podio-0.13-7tb2prisqr3n737rvemdeyxxc5xgwjhje/include/podio/EventStore.h(39): here cannot be referenced -- it is a deleted function
```

https://gitlab.cern.ch/ebrondol/clue/-/blob/debug_podio/src/CMakeLists.txt#L69-74

- Inputs are welcome!

- !!! For now, run only the C++ version of CLUE
Create simulated EDM4HEP events

- Simple simulated events created with [EDM4HEP test](EDM4HEP)
  - 10 single particle events
  - Producing CaloHitContribution & SimCalorimeterHit
  - Layer extracted with DD4HEP::DDSegmentation ([code](code))

```cpp
const BitFieldCoder bf("system:5,side:-2,layer:9,module:8,sensor:8,x:32:-16,y:-16");
auto layer = bf2.get_field("layer");
```

Consider to insert Layer info in EDM4HEP?

- More realistic events
  - Recently received a recipe to create EDM4HEP output w/CLIC reconstruction

¡¡¡ Soon updates
Conversion code EDM4HEP ↔ CSV

- Code in debug_podio branch
  - As already mentioned, run only the C++ version of CLUE
  - Change in main.cc to read both root and csv files and create output file name
    - Choice is done looking at the name at run time
  - Store x, y, layer, weights array from input file in read_events.h
Conclusions

- CLUE4HEP is starting being a reality
  - Compilation with CMake made our lives simpler
  - Still few open issues to fix (in particular CUDA vs PODIO compatibility)
  - C++ version of the repo works fine with (very!) simple EDM4HEP events
- Next step is looking into a more realistic cases with CLIC sim/reco (thanks!)
- When ready, the plan is to then move the repository to central key4hep

- Any feedback is welcome
Backup