CLUEstering

Simone Balducci[1, 2, 3] Felice Pantaleo[3] Marco Rovere[3] Aurora Perego[3, 4] Wahid Redjeb[3] Francesco Giacomini[2]

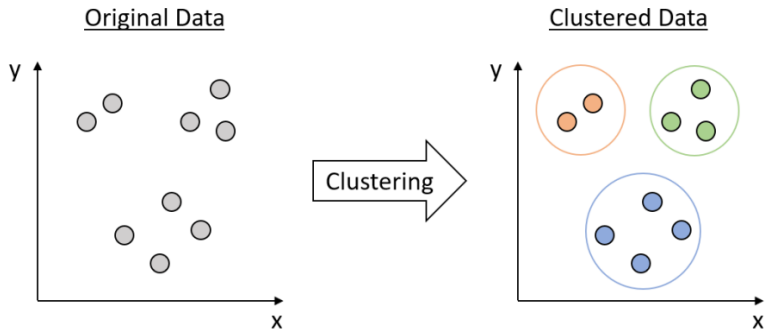
[1]University of Bologna [2]INFN-CNAF [3]CERN [4]University of Milano Bicocca

26/02/2026



Clustering

- Clustering is used to group data based on some measure of proximity or similarity



- It's a widely used technique because it allows to reconstruct classes of objects when there is no truth information available

The need for a density-based weighted clustering algorithm

- Density-based clustering finds clusters by indentifying regions with **high density of points.**
  - → the robustness to noise of such algorithms makes them particularly good for experimental applications

The need for a density-based weighted clustering algorithm

- Density-based clustering finds clusters by indentifying regions with **high density of points.**
  - → the robustness to noise of such algorithms makes them particularly good for experimental applications
- Weighted clustering allows to construct the clusters by considering a different weight for each point, representing their respective importance
  - → this makes them useful for applications where points are associated with some signal measures or a-priori knowledge

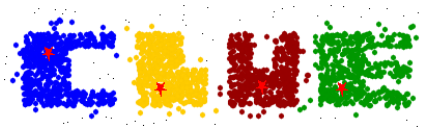
The need for a density-based weighted clustering algorithm

- Density-based clustering finds clusters by indentifying regions with **high density of points.**
  - → the robustness to noise of such algorithms makes them particularly good for experimental applications
- Weighted clustering allows to construct the clusters by considering a different weight for each point, representing their respective importance
  - → this makes them useful for applications where points are associated with some signal measures or a-priori knowledge
- Most density-based algorithms don't support weighted clustering out-of-the-box
  - → They need hand-made modifications to the dataset or to the distance matrix

The need for a density-based weighted clustering algorithm

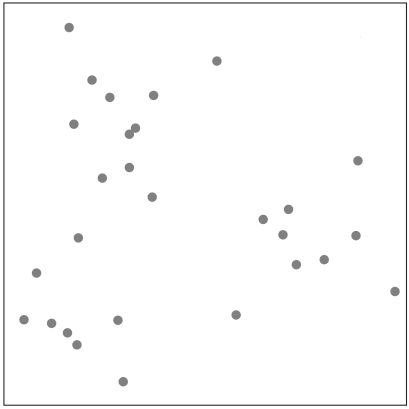
- Density-based clustering finds clusters by indentifying regions with **high density of points.**
  - → the robustness to noise of such algorithms makes them particularly good for experimental applications
- Weighted clustering allows to construct the clusters by considering a different weight for each point, representing their respective importance
  - → this makes them useful for applications where points are associated with some signal measures or a-priori knowledge
- Most density-based algorithms don't support weighted clustering out-of-the-box
  - → They need hand-made modifications to the dataset or to the distance matrix
- There is a need for an alternative solution that combines the power of **density-based** algorithms with the generality of **weighted** clustering

The CLUE algorithm



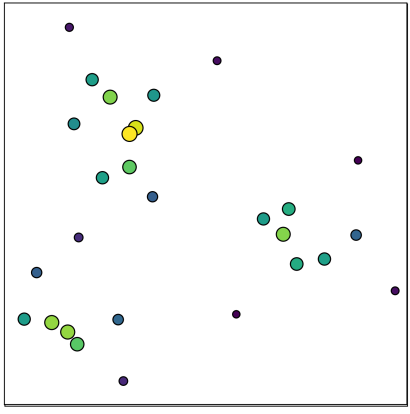
- CLUE (CLUstering of Energy) [1] is a density-based clustering algorithm used in the CMS experiment at LHC
- It was originally designed for the clustering of hits in the calorimeters
- Each point has a weight which is used when calculating the densities
- The weights are the energy deposit measurements of the detector layer sensors

Description of the algorithm



Description of the algorithm

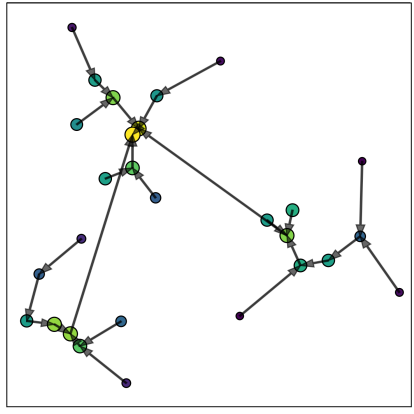
step 1 $\rightarrow$ Computation of the local density for each point



Description of the algorithm

step 1 $\rightarrow$ Computation of the local density for each point

step 2 $\rightarrow$ Selection of the *nearest highers*

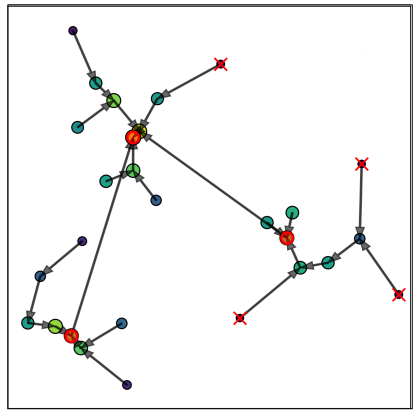


Description of the algorithm

step 1 $\rightarrow$ Computation of the local density for each point

step 2 $\rightarrow$ Selection of the *nearest highers*

step 3 $\rightarrow$ Finding clusters and outliers



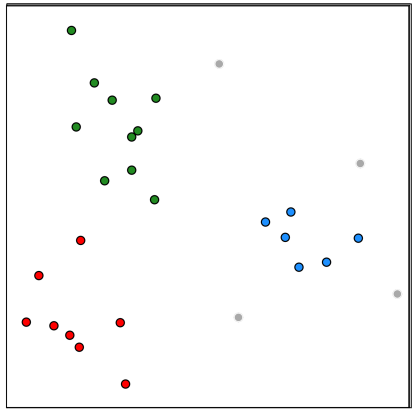
Description of the algorithm

step 1 $\rightarrow$ Computation of the local density for each point

step 2 $\rightarrow$ Selection of the *nearest highers*

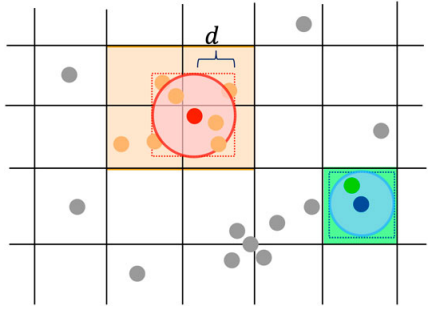
step 3 $\rightarrow$ Finding clusters and outliers

step 4 $\rightarrow$ Assigning hits to clusters



Optimizing neighbors query: Tiles

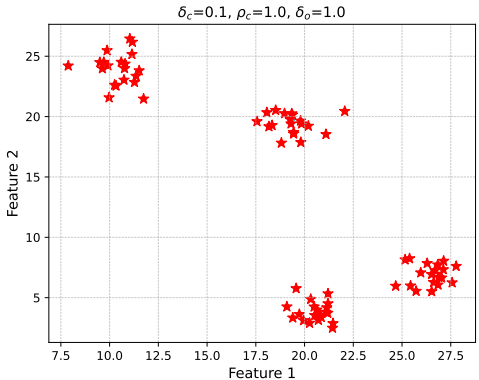
- If every point had to be compared with every other point, the resulting complexity would be $O(n^2)$
- This can be improved using a spatial indexing method
- The clustering space is divided in a grid of *tiles*
- Then, every point's potential neighbors are only the points in the neighboring tiles
  - → best-case complexity $O(n)$



The parameters of CLUE

3 parameters:

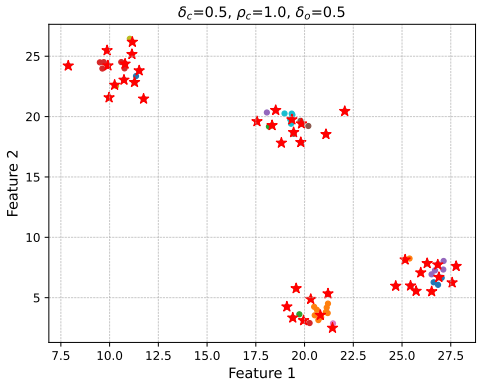
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

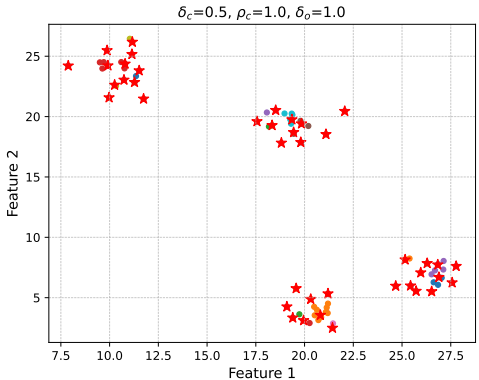
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

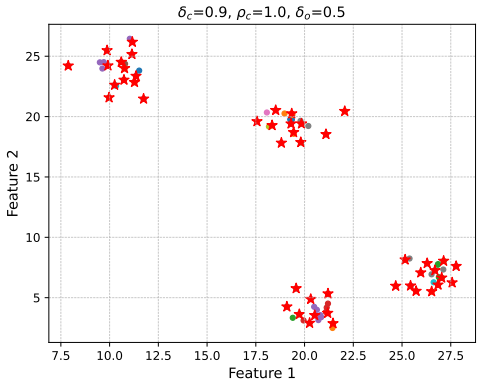
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

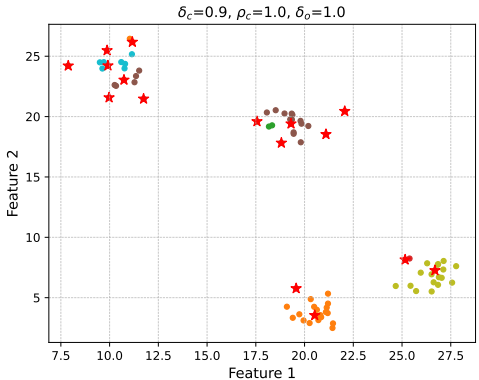
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

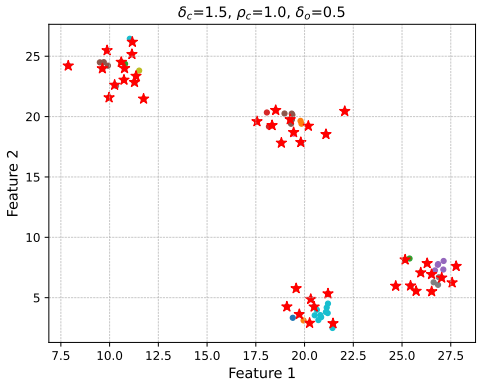
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

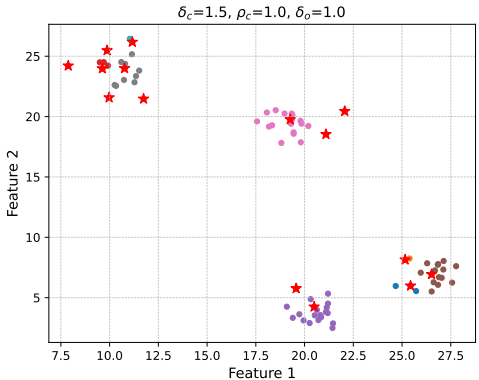
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

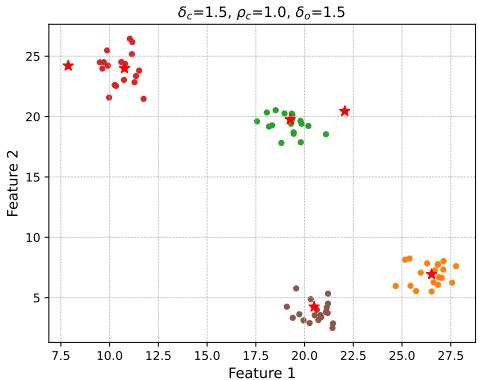
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

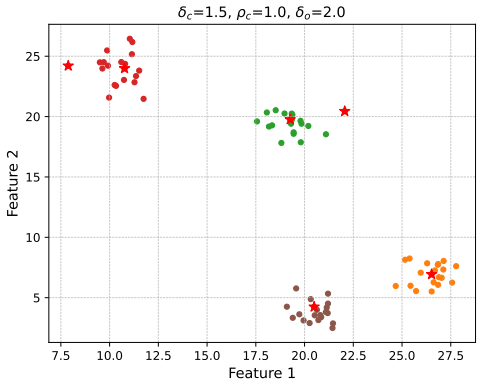
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

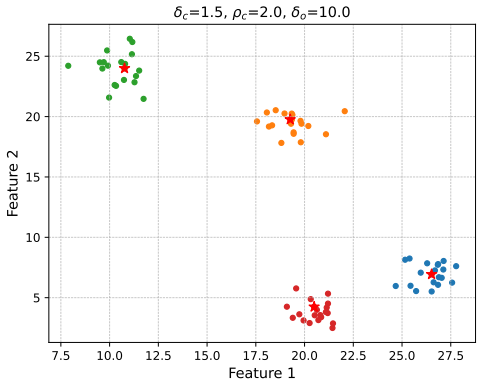
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

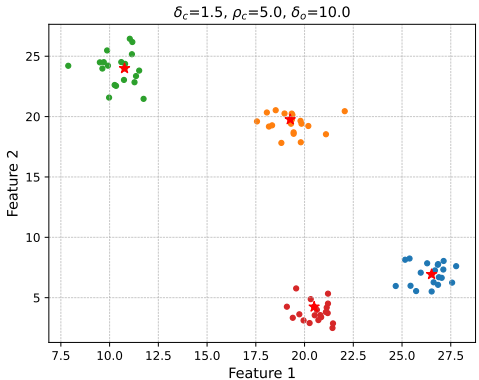
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

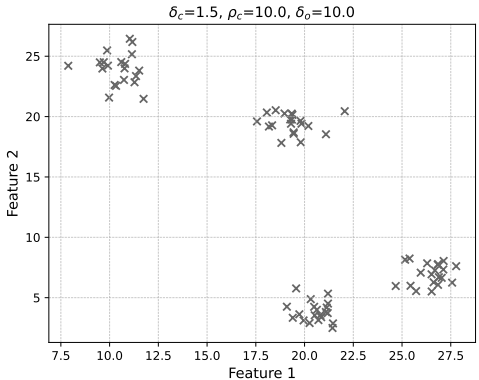
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



The parameters of CLUE

3 parameters:

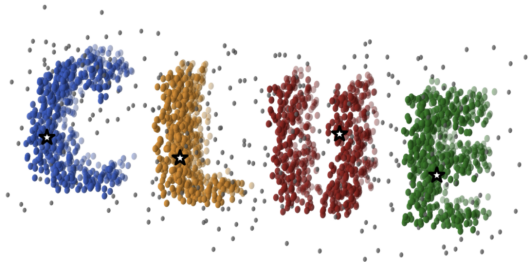
- density_radius (δ_c), size of query range for computation of local density
- `outlier_distance` (δ_o), size of query range for cluster extension
- `min_density` (ρ_c), density cut-off for promotion to cluster seed



From CLUE ...

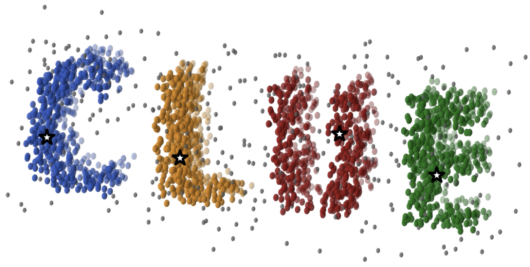
- CLUE was specifically tailored to work in the CMS detector
- 2-dimensional clustering for each of the layers
- Could not be used on a general dataset

From CLUE ... to CLUEstering



- CLUEstering [2][3][4] provides a generalization of CLUE
  - It's a general-purpose, header-only library
  - Applicable to any number of dimensions

From CLUE ... to CLUEstering



- CLUEstering [2][3][4] provides a generalization of CLUE
  - → It's a general-purpose, header-only library
  - → Applicable to any number of dimensions
- The backend and main interface are written in C++20. It's easily usable with CMake and installable with Conan
- It also provides a Python interface, which makes it easily usable by the scientific community

What can CLUEstering be used for?

- The implementation of CLUEstering makes it very general

What can CLUEstering be used for?

- The implementation of CLUEstering makes it very general
- It can be applied to a large variety of problems that use clustering
  - → in particular density-based clustering

What can CLUEstering be used for?

- The implementation of CLUEstering makes it very general
- It can be applied to a large variety of problems that use clustering
  - → in particular density-based clustering
- The main requirement is that data provides numerical coordinates

Performance portability in CLUEstering

- CLUE is a highly-parallel algorithm
- It's designed to work well on heterogeneous platforms
- The backend of CLUEstering is implemented with alpaka [5][6]
  - However, alpaka is hidden from the user. It only needs to be included in the final executable
- Users can run the clustering on any backend with a single command

C++ User interface: the Clusterer

- The Clusterer is the class containing the algorithm's logic
- It handles the allocation of the internal buffers
- It's templated on the number of dimensions and the coordinate's data type
- It contains the `make_clusters` method, which runs the algorithm on the device

```
1 clue::Clusterer<2> algo(queue, dc, rhoc, rm);
2
3 algo.make_clusters(queue, h_points);
```

C++ User interface: the PointsHost/Device containers

- `PointsHost/Device` are SoA[1] data structures used to handle the clustering data
- The columns of the SoA are the coordinates, weights and the cluster indexes
  - `PointsDevice` internally allocates extra columns only needed inside the kernels
- They can allocate the memory or use memory allocated externally
- Data can be copied from host to device with the `copyToHost` and `copyToDevice` functions
- They provide getters for the number of clusters and cluster-to-points associations

[1] Structure Of Array

C++ User interface: distance metrics

- The library provides several standard distance metrics
  - → Euclidian/weighted Euclinean, Manhattan, Chebyshev/weighted Chebyshev, periodic Euclidean
- Users can define metrics for theirs specific use-cases
- The call operator of the metric must be executable on both the host and the device
  - — `ALPAKA_FN_HOST_ACC`, `__host__ __device__`, ...
- The metric must also satisfy the `distance_metric` concept

```
template <typename TMetric, std::size_t Ndim>
concept distance_metric = requires(TMetric&& metric) {
  {
    metric(std::array<typename TMetric::value_type, Ndim + 1>{},
           std::array<typename TMetric::value_type, Ndim + 1>{})
  } -> std::same_as<typename TMetric::value_type>;
};
```

The Python interface

- The Python interface is very similar to the C++ one, being centered around the `clusterer` class
- Data can be read and saved in SoA format with the `read_data` method
  - the method takes data in different formats like numpy arrays, lists and pandas dataframes
- The clustering is run with the `run_clue` or `fit/fit_predict` methods
  - the alpaka backend can be selected by passing a string ("cpu serial", "cpu tbb", "gpu cuda", ...)
- The clusterer also provides methods for saving the results to a csv file or to plot the clustered data

How to use the library (C++ interface)

Both the C++ and the Python interfaces of the library are straightforward to use

```cpp
#include <CLUEstering/CLUEstering.hpp>

int main() {
  auto queue = clue::get_queue(0u);

  clue::PointsHost<Ndim> h_points = clue::read_csv<Ndim>(queue, "data.csv");
  clue::PointsDevice<Ndim> d_points(queue, h_points.size());

  const auto density_radius = 20.f, min_density = 10.f;
  clue::Clusterer<Ndim> algo(queue, density_radius, min_density);

  algo.make_clusters(queue, h_points, d_points);
  auto cluster_ids = h_points.clusterIndexes();
  auto seed_map = h_points.isSeed();
}
```

How to use the library (Python interface)

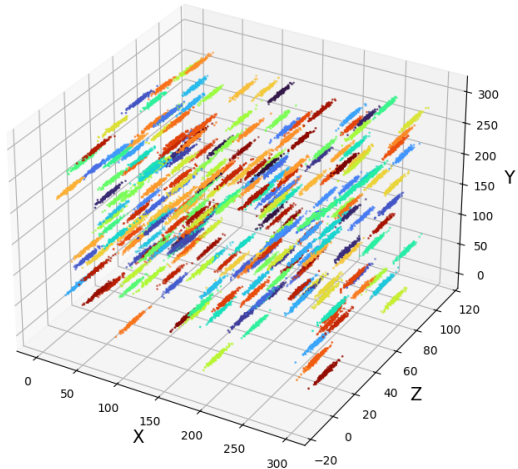
Both the C++ and the Python interfaces of the library are straightforward to use

```
import CLUEstering as clue

density_radius = 20.
min_density = 10.
clust = clue.clusterer(density_radius, min_density)
clust.read_data(data)
clust.run_clue()
clust.to_csv('./output/', 'data_results.csv')
```

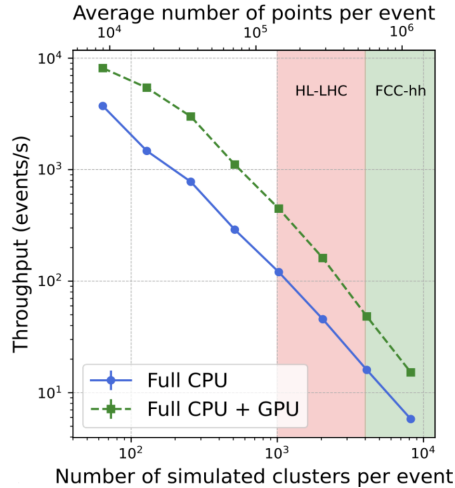
An example dataset

- This dataset is representative of a hit reconstruction in an experiment at LHC
- Multiple datasets are generated increasing the total number of clusters
- The clusters are generated in a volume of $3 \times 3 \times 1$ meters, which mimics the endcap region of the CMS detector

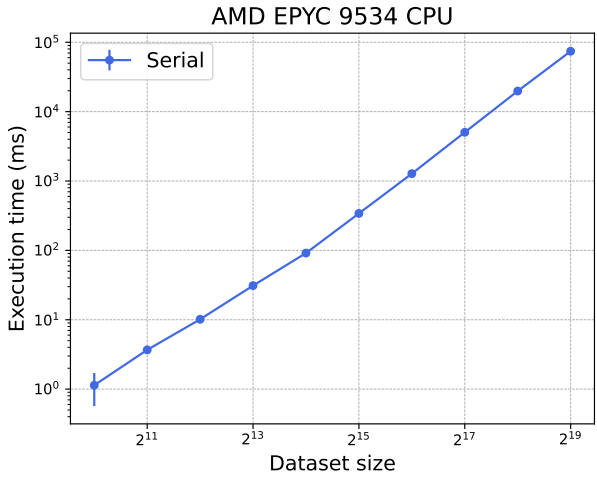


Throughput measures in multithreaded execution

- Modern HEP software frameworks often are highly multithreaded in order to maximize the number of events processed concurrently
- The benchmark on the right shows the throughput obtained by fully occupying a machine with an AMD EPYC 9754 CPU and an NVIDIA Tesla T4 GPU, as a function of the number of clusters reconstructed

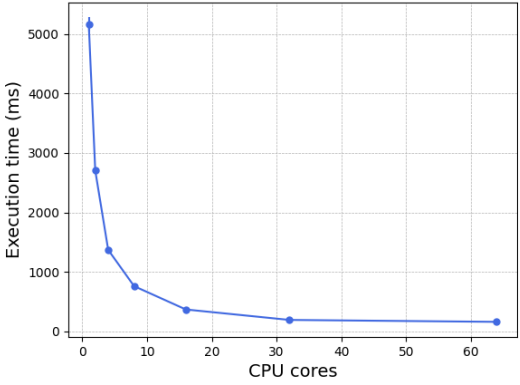


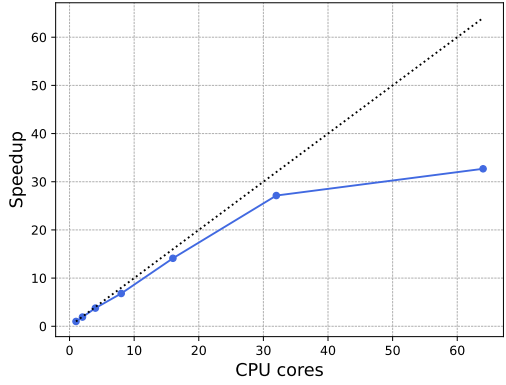
Scaling of the serial execution



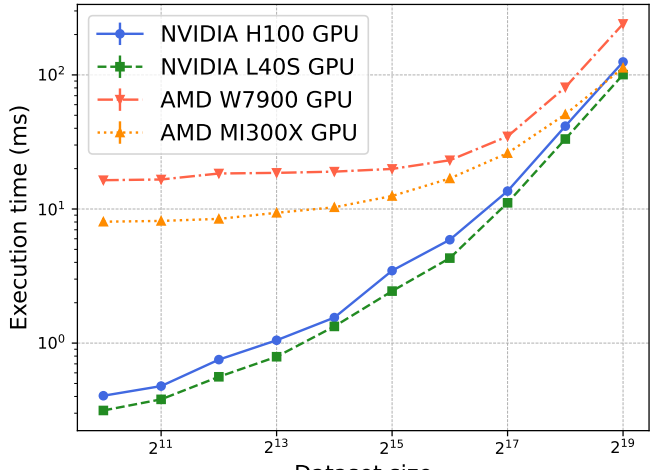
Scaling of OpenMP backend with CPU cores

These benchmarks were performed on an AMD EPYC 9534 CPU with 64 cores





Comparison of different GPU models



Current applications in CMS software

We are currently investigating the use of CLUEstering in different parts of the CMS software:

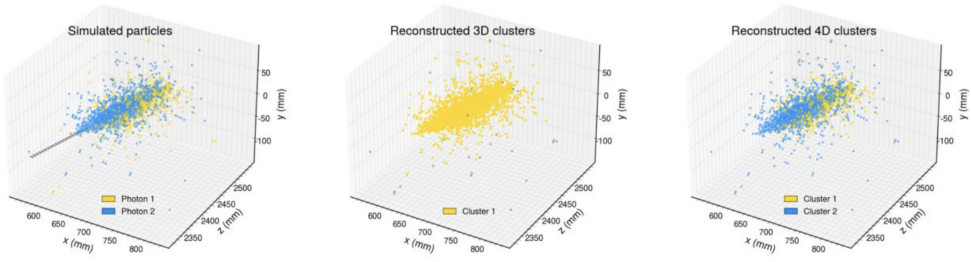
- Pixel vertex reconstruction in the HLT
- Reconstruction of low p_T τ jets in L1 scouting
- Layer clusters and trackster reconstruction in the TICL framework for the HGCal calorimeter

τ jet reconstruction in L1 scouting

- In this application, each event contains between 300 and 400 points, and 3564 events are grouped and processed together, making up *orbits*
- Processing these events singularly on the GPU is extremely inefficcient, as it doens't make full use of the accelerator's capabilities
- Thus, in CLUEstering we developed an interface for *batched clustering*, which processes the entire orbit calling the kernels only once
  - → this granted a $\times$100 speed-up with respect to clustering the events singularly

Developments outside CMS: FCC

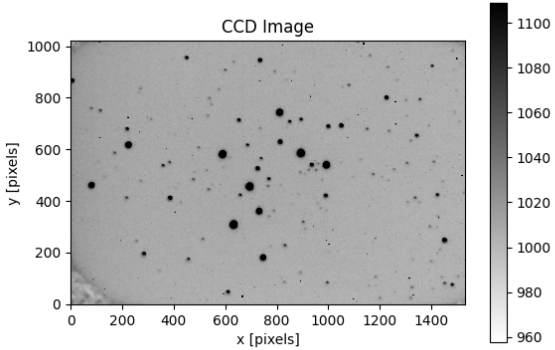
- We are currently investigating applications in the FCC software in the context of the CERN Experimental Physics R&D program
- CLUEstering's ability for N-dimensional clustering allows to use the time information
  - → the plot below shows the reconstruction of two simulated particles in a $4D$ space[2]



[2] credit: Aurora Perego (University of Milano Bicocca, CERN)

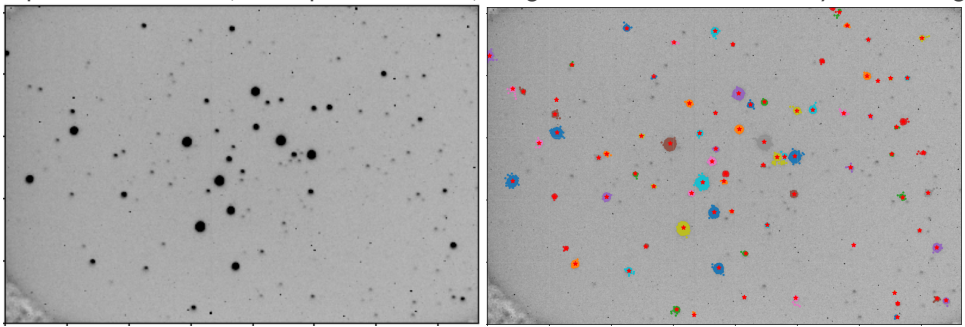
Star detection in astronomy

- Modern telescopes use CCDs (charge-coupled devices) to convert impinging photons into electrons
- Each pixel contains the number of electrons



CLUEstering for star detection

Comparison of the PSF (Point Spread Function) image and the stars detected by CLUEstering



Future plans

- Investigating the possibility of using Fourier transform to optimize the computation of the points' local density
- Supporting the use of externally-defined allocators, in particular **caching allocators**
- Support GPU containers as input for the Python interface, like **cupy arrays** and **cuda dataframes**
- Improve the input and output interface for common data formats
  - CSV, HDF5, ROOT, …
- Implement a parameter auto-tuning functionality for getting the optimal combination of parameters
  - this can be done with the library **patatune**

References

[1] Marco Rovere et al. “CLUE: A Fast Parallel Clustering Algorithm for High Granularity Calorimeters in High-Energy Physics”. In: *Frontiers in Big Data* Volume 3 - 2020 (2020). ISSN: 2624-909X. DOI: 10.3389/fdata.2020.591315.

[2] URL: https://gitlab.cern.ch/kalos/CLUEstering.

[3] URL: https://github.com/cms-patatrack/CLUEstering.

[4] URL: https://cms-patatrack.github.io/CLUEstering.

[5] Erik Zenker et al. “Alpaka - An Abstraction Library for Parallel Kernel Acceleration”. In: IEEE Computer Society, May 2016. arXiv: 1602.08477. URL: http://arxiv.org/abs/1602.08477.

[6] URL: https://github.com/alpaka-group/alpaka.

Thanks for the attention!
Questions?