The 24th International Conference on Computing in High Energy and Nuclear Physics
CHEP2019, Adelaide, Australia. Nov 04 - 08, 2019

GPU-based Clustering Algorithm for the CMS High Granularity Calorimeter

*Z. Chen[1], A. Di Pilato[2], F. Pantaleo[3], M. Rovere[3]

On behalf of the CMS Collaboration

[1] Northwestern University, [2] University of Bari & INFN, [3] CERN

Luminosity Forecast

● Peak luminosity —Integrated luminosity

Ultimate HL-LHC Schedules and Luminosity Forecasts [1]

HL-LHC

CMS Phase1 upgrade

CMS Phase2 upgrade

LS1 LS2 LS3 LS4 LS5

now

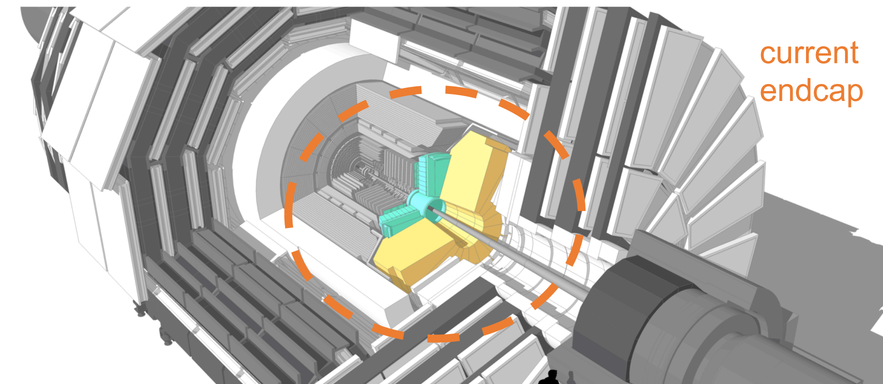
Designed limit of current endcap L=500/fb

Luminosity [cm^{-2}s^{-1}]: 0.0E+00, 1.0E+34, 2.0E+34, 3.0E+34, 4.0E+34, 5.0E+34, 6.0E+34, 7.0E+34, 8.0E+34

Integrated luminosity [fb^{-1}]: 0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000

Year: 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38

- After 2026, HL-LHC will deliver ultimate luminosity up-to $7.5\times 10^{34}\ cm^{-2}s^{-1}$ and produce high pileup events up-to PU200. [1]
- Current CMS endcap calorimeters (ECAL and HCAL) are designed for a 500/fb lifetime integrated luminosity.[2] They need to be upgraded during CMS phase-II upgrade in LS3 before reaching their radiation limit.



[1] https://lhc-commissioning.web.cern.ch/lhc-commissioning/schedule/HL-LHC-plots.htm

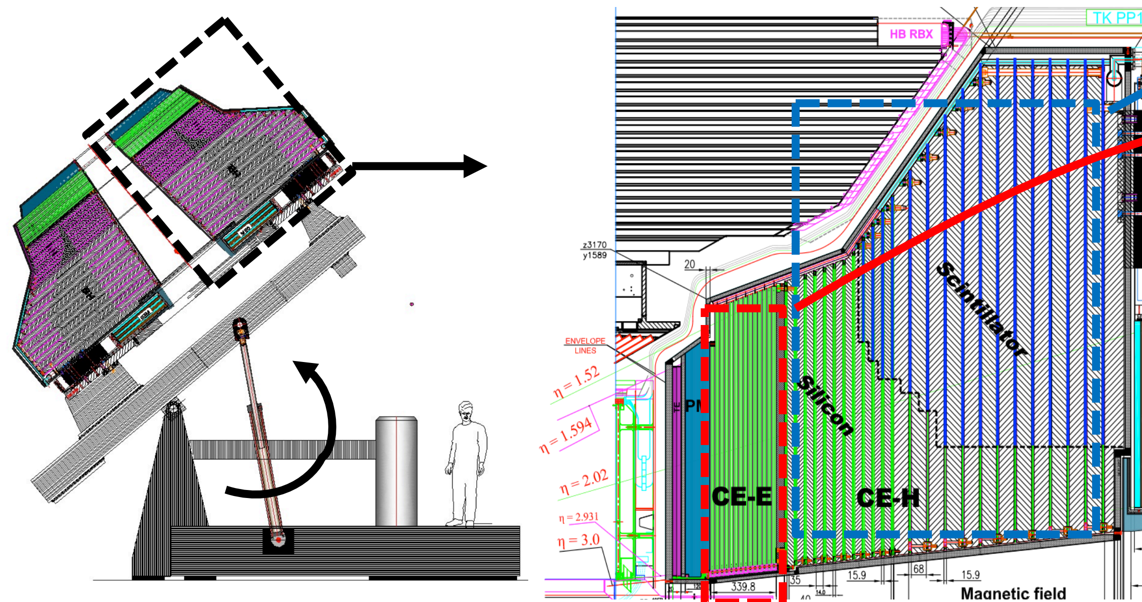
[2] CMS Collaboration, "Technical Proposal for the Phase-II Upgrade of the Compact Muon Solenoid", Technical Report CERN-LHCC-2015-010, LHCC-P-008, 2015.

CMS High Granularity Calorimeter

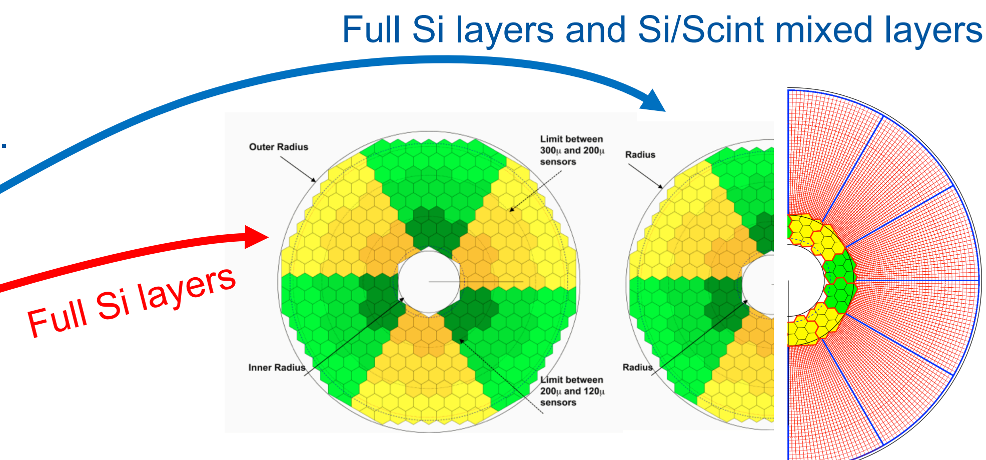
- Current CMS endcap ECAL and HCAL calorimeters will be replaced by High Granularity Calorimeter (HGCAL), a sampling calorimeter system based on Si sensors and plastic scintillators, during CMS phase-II upgrade.

  - Full system operates at $-35°\mathrm{C}$ maintained by a CO_2 cooling system
  - Covers $1.5 < |\eta| < 3.0$ on both left and right sides
  - Total size z=2m, r = 2.3m. Total weight 215 ton per endcap
  - 620 m^2 of Silicon sensors (120/200/300 um). 0.5-1.0 cm^2 → 6M channels
  - 400 m^2 of plastic scintillators with SiPM readout. 4-30 cm^2 → 240k channels.

- Plots from "The Phase-2 Upgrade of the CMS endcap calorimeter Technical Design Report" http://cds.cern.ch/record/2293646/files/

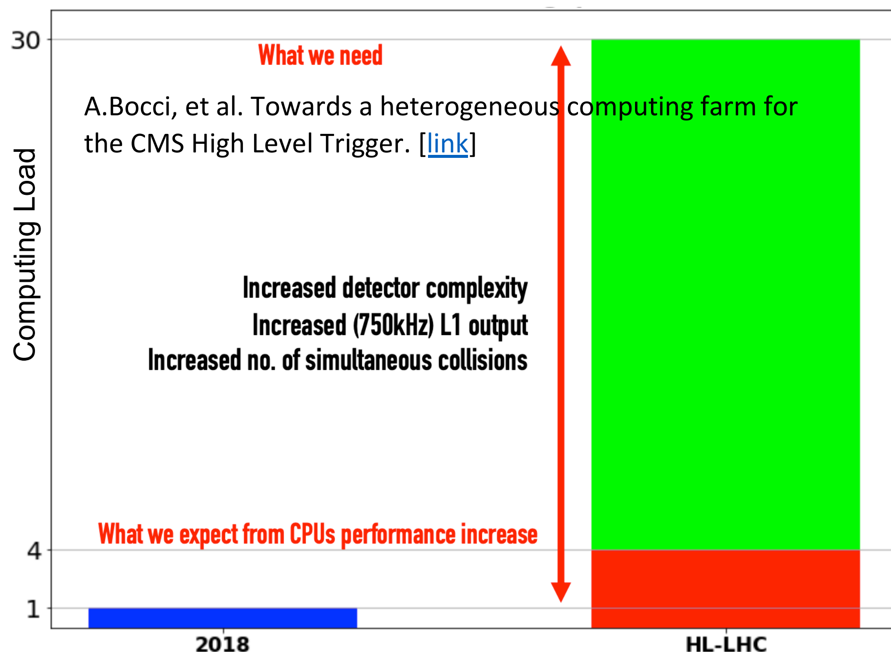


Full Si layers and Si/Scint mixed layers



	CE-E	CE-H
Number of layers	28 layer/endcap	8+14 layer/endcap
Sensor type	Si	Si, Scintillator
Absorber type	Cu, CuW, Pb	Stainless Steel, Cu
Thickness	$25X_0$, 1.3λ	$\sim 9.5\lambda$

Challenge of Computing



- CMS uses two-level trigger system, L1 Trigger and High Level Trigger (HLT), to reduce data rate from 400 MHz (LHC) → 100 kHz (CMS L1T output rate) →1 kHz (HLT output rate)
  - L1 Trigger: based on ASICs and FPGAs. Make decision in 4 us.
  - HLT: based on CPUs. Make decision in 300 ms.

- HLT in era of HL-LHC expects 30x more computing load
  - 4x from increased event complexity: upgraded detectors (~1.3x), higher pile-up (~3x)
  - 7.5x from increased event rate: L1 output rate 100 kHz → 750 kHz.

- Within 30x increased computing load, increase of CPU performance can account for only 4x. CPU alone is not enough to handle this computing challenge.

- It is particularly a huge challenge for HGCAL reconstruction to achieve the HLT time budget (< 20-50 ms *) for PU200 events in HL-LHC.

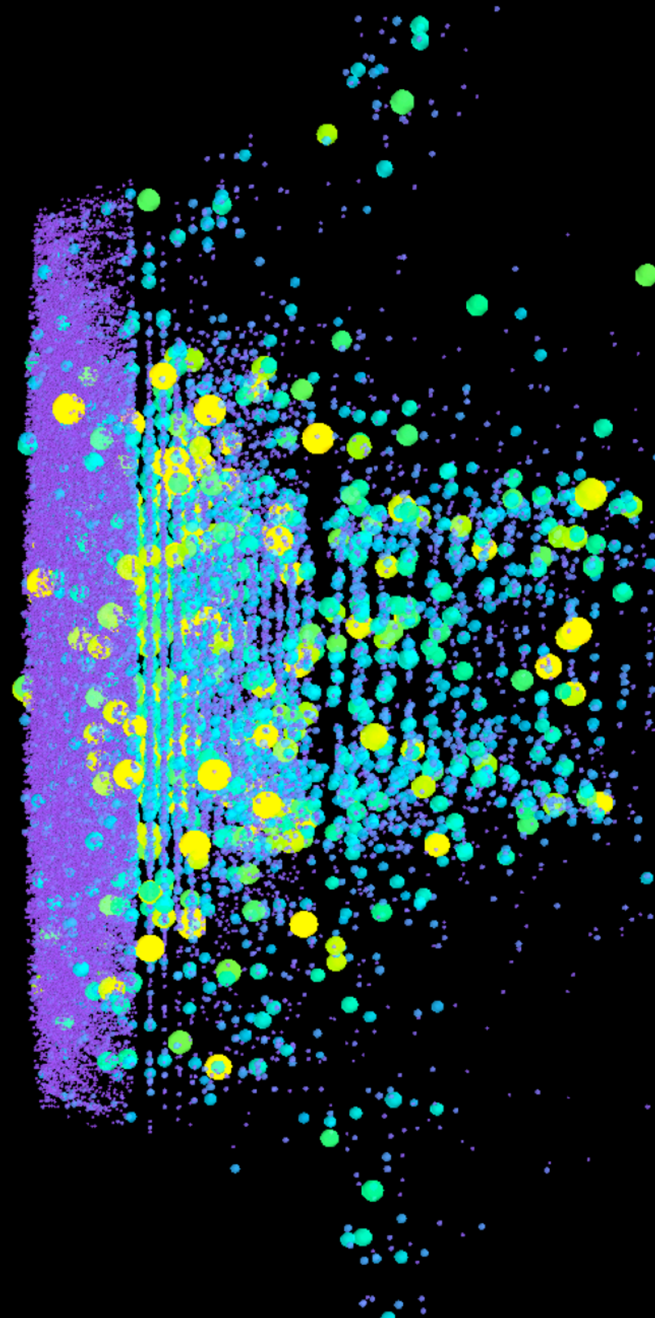
- GPU could be a solution. CUDA, an application programming interface (API) for General-Purpose computing on Graphics Processing Units (GPGPU), makes it possible to accelerate HGCAL and other HLT reconstruction with GPUs.

- 300 ms/event $\div \frac{7.5}{4(2)}$ x 33% = 53(27) ms/event

This estimating calculation just sets a scale of upper limit of HGCAL budget in HLT, where 300 ms is current total HLT budget, 7.5 is factor for increase of event rate, 4(2) is factor for optimistic(realistic) increase of # CPU threads., 33% is a conservative upper limit of HGCAL's portion of HLT online time.

Clustering in HGCAL

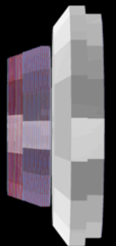
- Hits of a PU200 event in HGCAL. Color and size represent hit energy. Interaction point is on the left side.
- n~ O(100,000) hits
- HGCAL reconstruction starts by reconstructing 2D clusters layer-by-layer. k~O(10,000) clusters.
- Since cells are small compared to shower lateral size, an "energy density" is defined to better hint regional energy blobs.
- 3D showers are reconstructed by collecting and associating 2D layer clusters



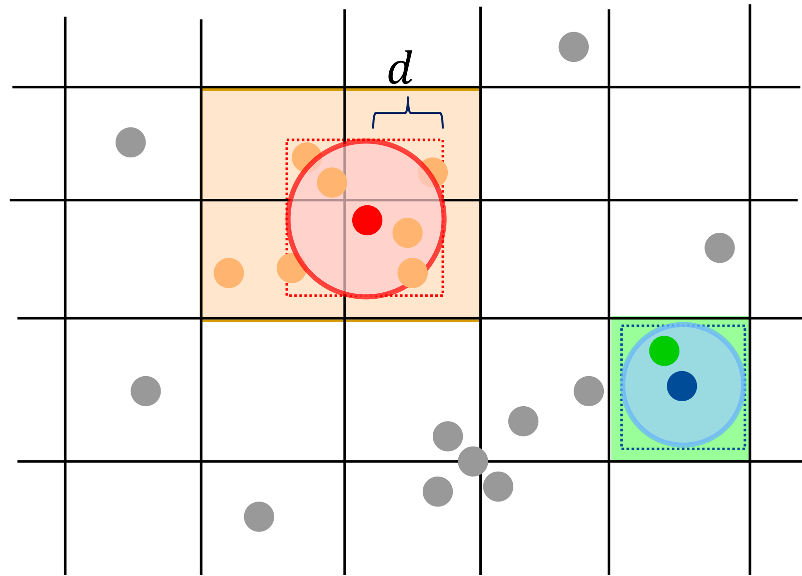
Features of HGCAL clustering task

$n > k \gg \frac{n}{k}$ in 2D

Fast and GPU-friendly



Clustering by Energy (CLUE) on GPU



Querying neighborhood N_d is one of the most frequent operations in density-based clustering algorithms. So need fast N_d query.

d-searchBox Ω_d

$$\Omega_d(i) = \{j : j \in \text{tiles touched by square window } (x_i \pm d, y_i \pm d)\}$$

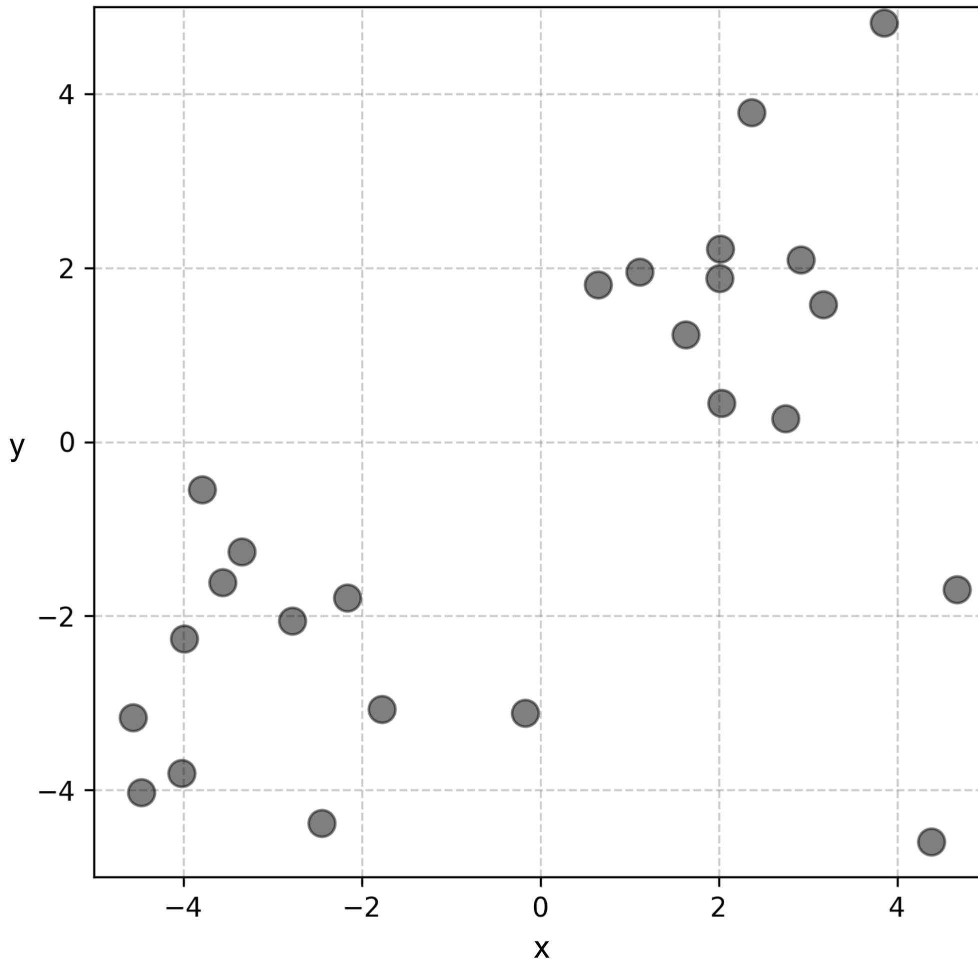
d-neighborhood N_d

$$N_d(i) = \{j : d_{ij} < d\} \subset \Omega_d(i)$$

To query N_d, we only need to loop of hits in Ω_d

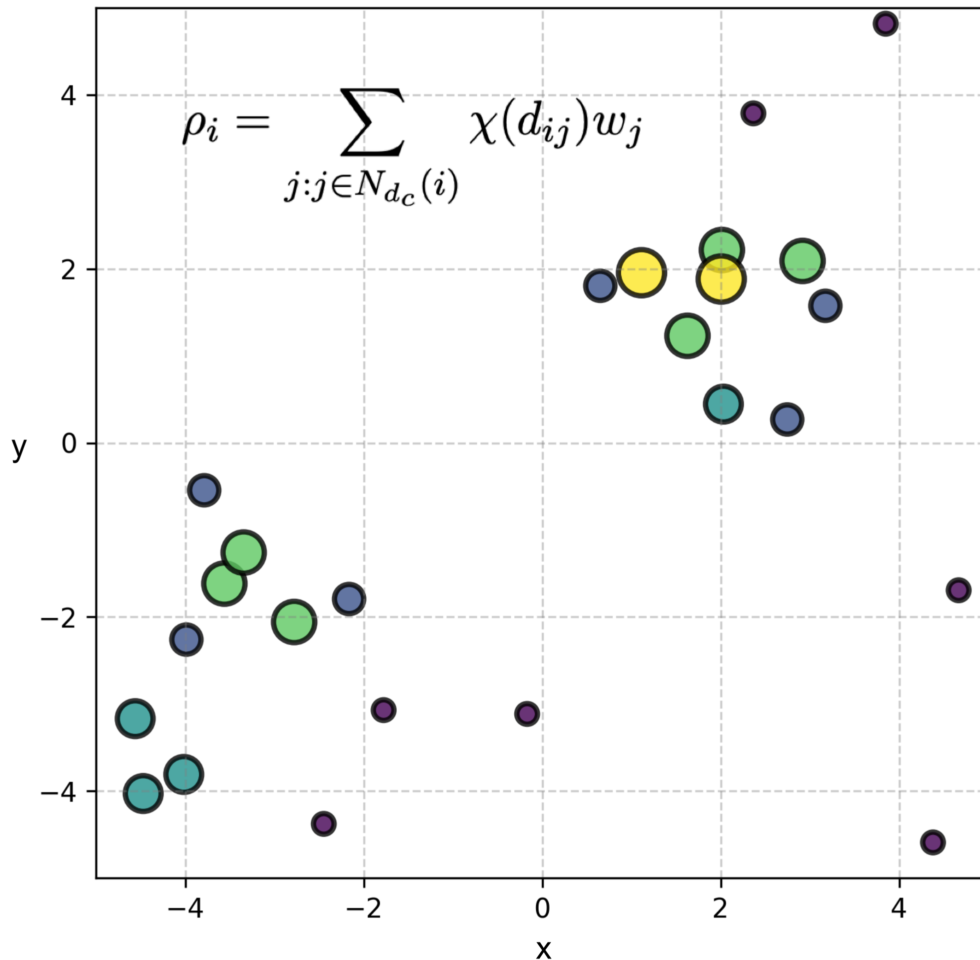
- Build "Grid Spatial Index" for hits on each layer
  - Grid tiles are small comparing with the size of HGCal layer
  - Each tile in the grid hosts indices of hits inside it and has a fix length of memory to store the hosted indices.
  - Points inside a tile can be directly accessed.
- Complexity of query d-neighborhood is O(1), given that d is small.
- Building spatial index is highly parallelizable on GPU.

Clustering by Energy (CLUE) on GPU



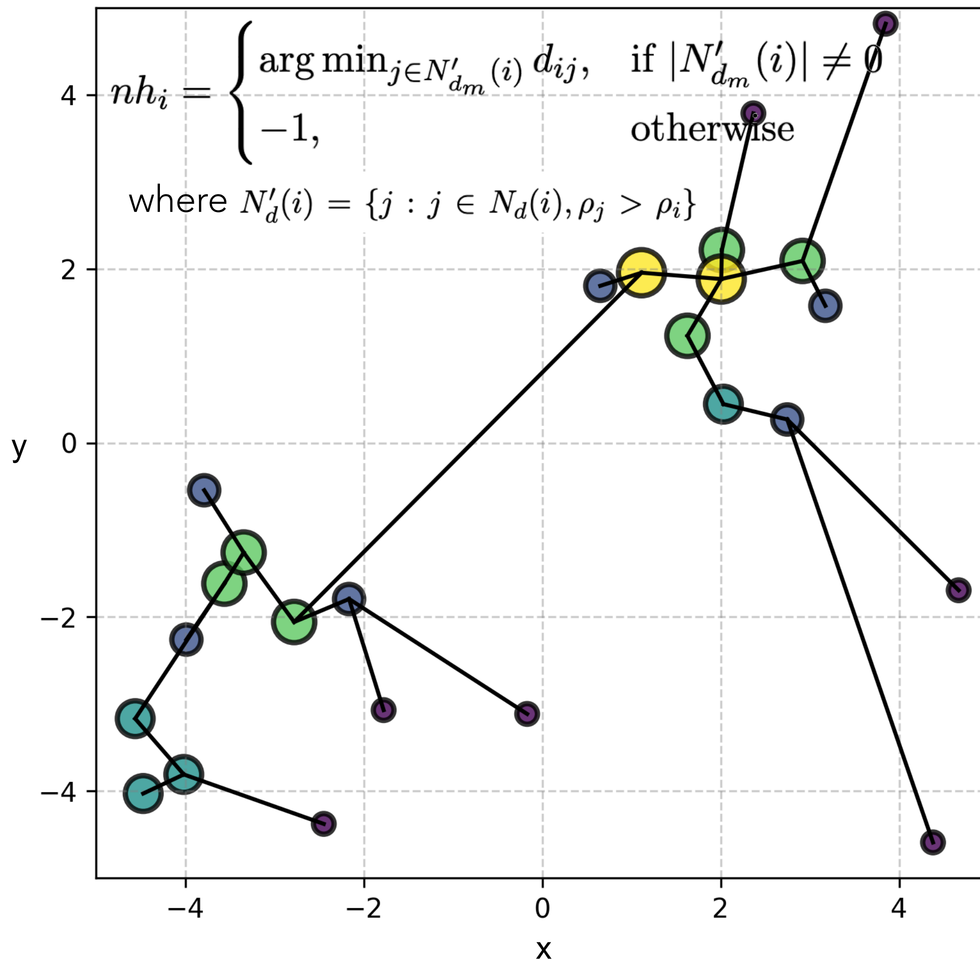
- **Step 0: Build Spatial Index**
  - **1 CUDA thread for each hit**
  - **Register the index of each point to corresponding tile**
- Step 1: Calculate Local density
  - 1 CUDA thread for each hit
  - Density defined on the left
- Step 2: Calculate Nearest Higher
  - 1 CUDA thread for each hit
  - Define $d_m \equiv \max(\boldsymbol{\delta_s}, \boldsymbol{\delta_o})$, where $\boldsymbol{\delta_s}, \boldsymbol{\delta_o}$ are algorithm parameters for seed promotion and outlier demotion
  - Within $N_{dm}(i)$, find the nearest points with higher density.
  - Calculate $\delta_i = dist(i, nh_i)$
- Step 3: Promote Seeds and Demote Outliers
  - 1 CUDA thread for each hit
  - Promote hit as seed if $\rho_i > \boldsymbol{\rho_c}, \delta_i > \boldsymbol{\delta_s}$
  - Demote hit as outlier if $\rho_i < \boldsymbol{\rho_c}, \delta_i > \boldsymbol{\delta_o}$
- Step 4: Assign Cluster ID
  - 1 CUDA thread for each seed
  - Push down the cluster ID from seeds through reversed chains of nearest higher

Clustering by Energy (CLUE) on GPU



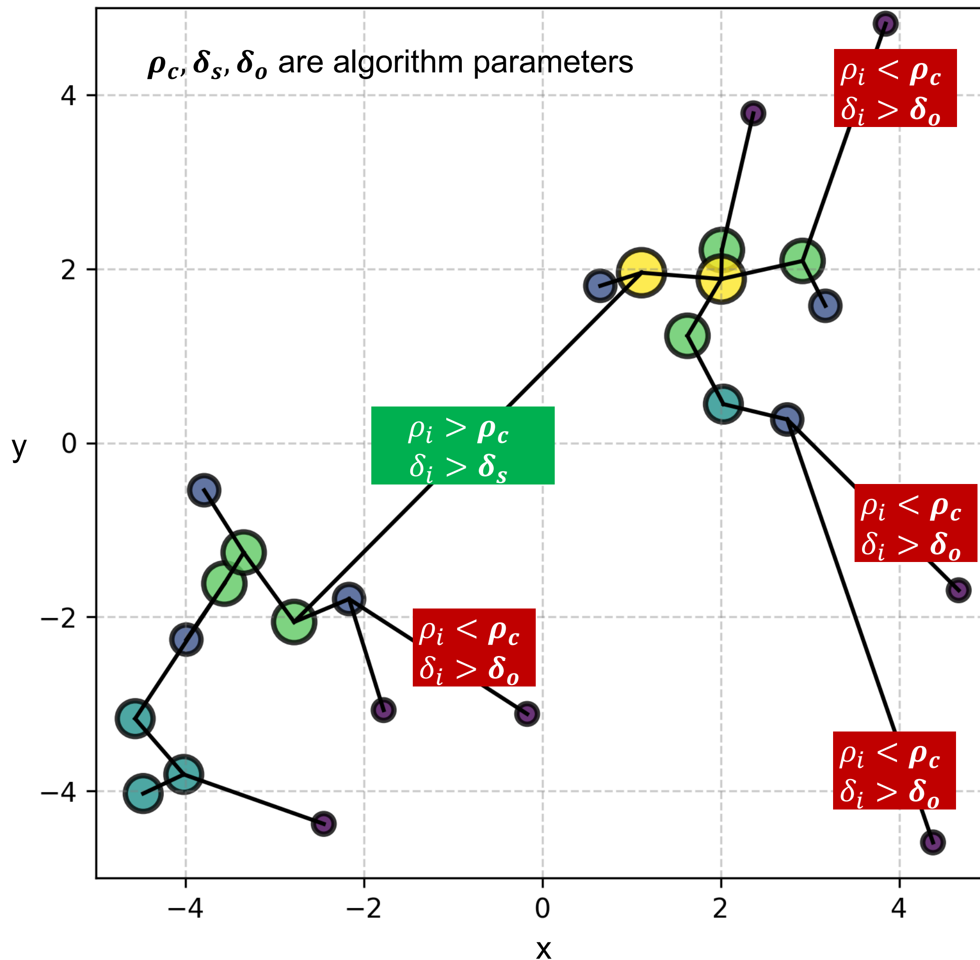
- Step 0: Build Spatial Index
  - 1 CUDA thread for each hit
  - Register the index of each point to corresponding tile
- **Step 1: Calculate Local Density**
  - **1 CUDA thread for each hit**
  - **Density defined on the left**
- Step 2: Calculate Nearest Higher
  - 1 CUDA thread for each hit
  - Define $d_m \equiv \max(\boldsymbol{\delta_s}, \boldsymbol{\delta_o})$, where $\boldsymbol{\delta_s}, \boldsymbol{\delta_o}$ are algorithm parameters for seed promotion and outlier demotion
  - Within $N_{dm}(i)$, find the nearest points with higher density.
  - Calculate $\delta_i = dist(i, nh_i)$
- Step 3: Promote Seeds and Demote Outliers
  - 1 CUDA thread for each hit
  - Promote hit as seed if $\rho_i > \boldsymbol{\rho_c}, \delta_i > \boldsymbol{\delta_s}$
  - Demote hit as outlier if $\rho_i < \boldsymbol{\rho_c}, \delta_i > \boldsymbol{\delta_o}$
- Step 4: Assign Cluster ID
  - 1 CUDA thread for each seed
  - Push down the cluster ID from seeds through reversed chains of nearest higher

Clustering by Energy (CLUE) on GPU



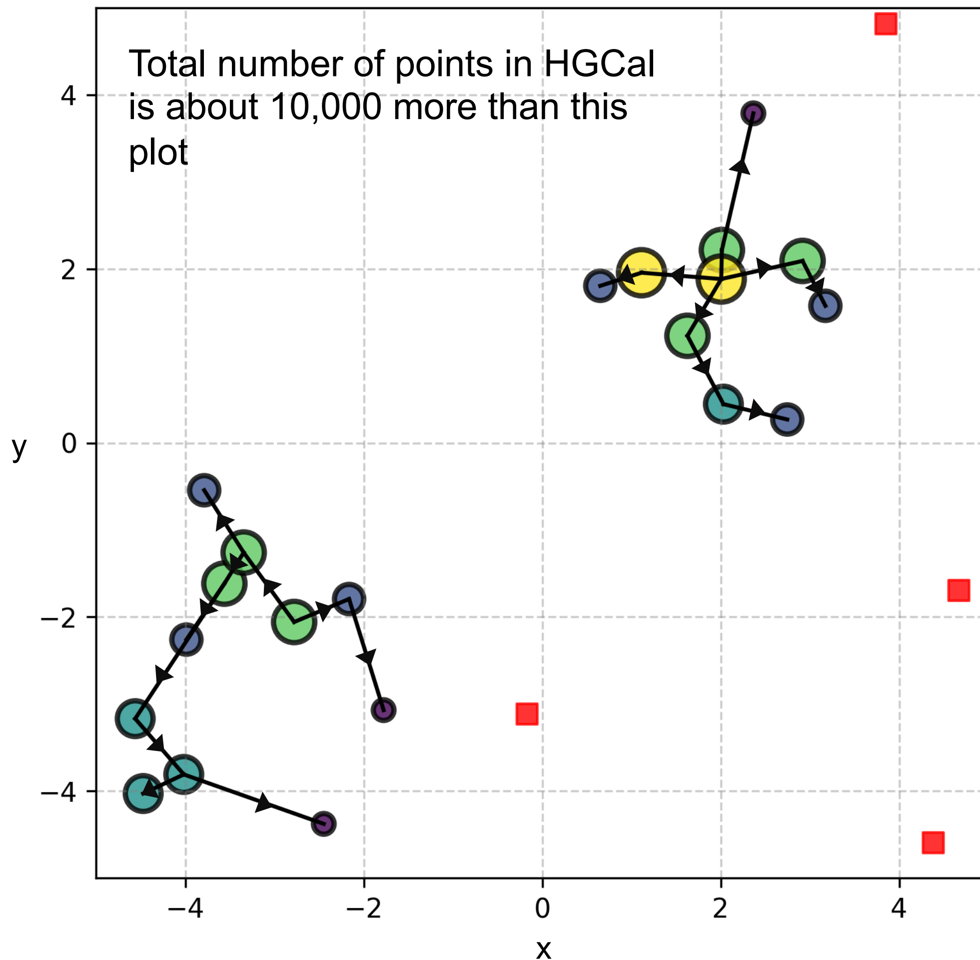
- ❖ Step 0: Build Spatial Index
  - 1 CUDA thread for each hit
  - Register the index of each point to corresponding tile
- ❖ Step 1: Calculate Local density
  - 1 CUDA thread for each hit
  - Density defined on the left
- ❖ **Step 2: Calculate "Nearest Higher"**
  - **1 CUDA thread for each hit**
  - **Define $d_m \equiv \max(\delta_s, \delta_o)$, where δ_s, δ_o are algorithm parameters for seed promotion and outlier demotion**
  - **Within $N_{dm}(i)$, find the nearest point with higher density.**
  - **Calculate $\delta_i = dist(i, nh_i)$**
- ❖ Step 3: Promote Seeds and Demote Outliers
  - 1 CUDA thread for each hit
  - Promote hit as seed if $\rho_i > \rho_c, \delta_i > \delta_s$
  - Demote hit as outlier if $\rho_i < \rho_c, \delta_i > \delta_o$
- ❖ Step 4: Assign Cluster ID
  - 1 CUDA thread for each seed
  - Push down the cluster ID from seeds through reversed chains of nearest higher

Clustering by Energy (CLUE) on GPU



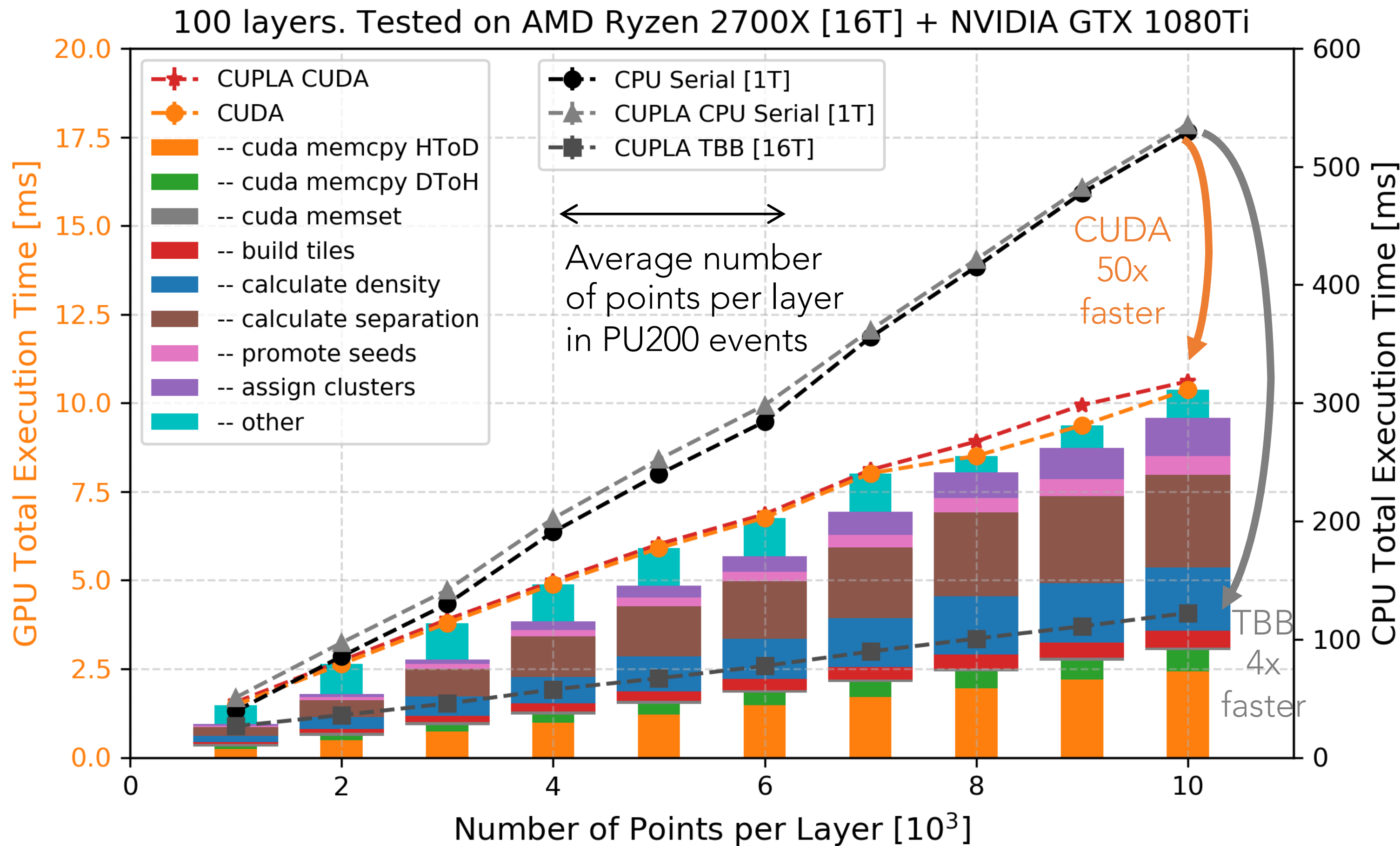
- **Step 0: Build Spatial Index**
  - 1 CUDA thread for each hit
  - Register the index of each point to corresponding tile
- **Step 1: Calculate Local density**
  - 1 CUDA thread for each hit
  - Density defined on the left
- **Step 2: Calculate Nearest Higher**
  - 1 CUDA thread for each hit
  - Define $d_m \equiv \max(\delta_s, \delta_o)$, where δ_s, δ_o are algorithm parameters for seed promotion and outlier demotion
  - Within $N_{dm}(i)$, find the nearest points j with higher density.
  - Calculate $\delta_i = dist(i, nh_i)$
- **Step 3: Promote Seeds and Demote Outliers**
  - **1 CUDA thread for each hit**
  - **Promote hit as seed if $\rho_i > \rho_c, \delta_i > \delta_s$**
  - **Demote hit as outlier if $\rho_i < \rho_c, \delta_i > \delta_o$**
- **Step 4: Assign Cluster ID**
  - 1 CUDA thread for each seed
  - Push down the cluster ID from seeds through reversed chains of nearest higher

Clustering by Energy (CLUE) on GPU



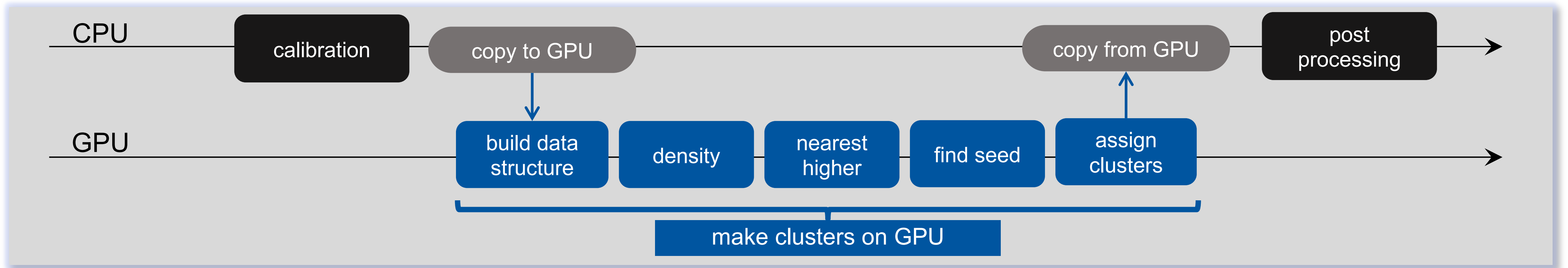
- Step 0: Build Spatial Index
  - 1 CUDA thread for each hit
  - Register the index of each point to corresponding tile
- Step 1: Calculate Local density
  - 1 CUDA thread for each hit
  - Density defined on the left
- Step 2: Calculate Nearest Higher
  - 1 CUDA thread for each hit
  - Define $d_m \equiv \max(\boldsymbol{\delta_s}, \boldsymbol{\delta_o})$, where $\boldsymbol{\delta_s}, \boldsymbol{\delta_o}$ are algorithm parameters for seed promotion and outlier demotion
  - Within $N_{dm}(i)$, find the nearest points j with higher density.
  - Calculate $\delta_i = dist(i, nh_i)$
- Step 3: Promote Seeds and Demote Outliers
  - 1 CUDA thread for each hit
  - Promote hit as seed if $\rho_i > \boldsymbol{\rho_c}, \delta_i > \boldsymbol{\delta_s}$
  - Demote hit as outlier if $\rho_i < \boldsymbol{\rho_c}, \delta_i > \boldsymbol{\delta_o}$
- **Step 4: Assign Cluster ID**
  - **1 CUDA thread for each seed**
  - **Push the cluster ID from seeds to other hits through the reversed chains of nearest higher.**

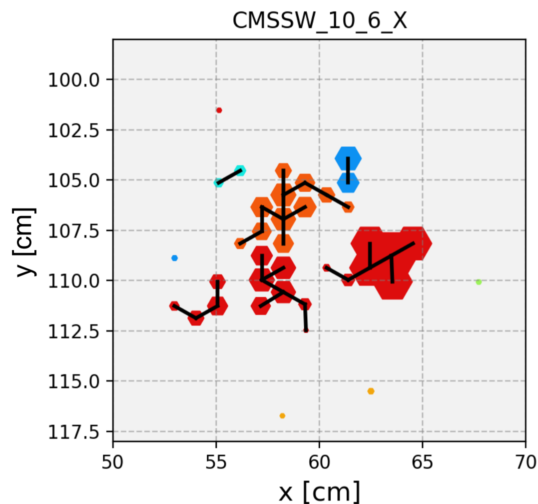
Clustering by Energy (CLUE) on GPU



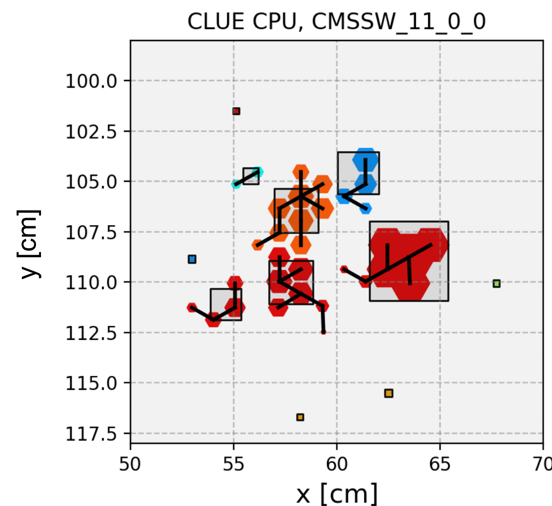
- Test standalone CLUE implemented with C++, CUDA and CUPLA.
- CUPLA is a wrapper of Alpaka to conveniently port CUDA code to other accelerator backends, such as Intel Thread Building Block (TBB)
- In the test, each event has 100 layers. On each layer, generate a total of n random 2D points from $\frac{n}{10}$ 2D gaussian distributions, each gaussian giving 10 points.
- AMD Ryzen 2700 + NVIDIA GTX 1080Ti.
- Total run time is O(n)
- TBB (16T) is ~4x faster than serial CPU.
- CUDA is ~50x faster than serial CPU.
- Speed up factors depend on CPU and GPU. Intel Xeon Silver + NVIDIA Tesla V100 is in backup.

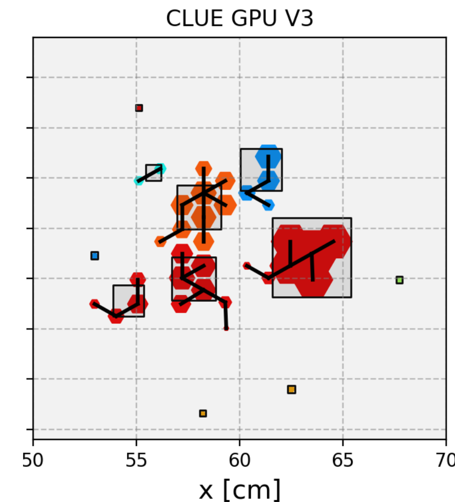
CLUE in CMS Software (CMSSW) Framework





Previous clustering in CMSSW framework [1]



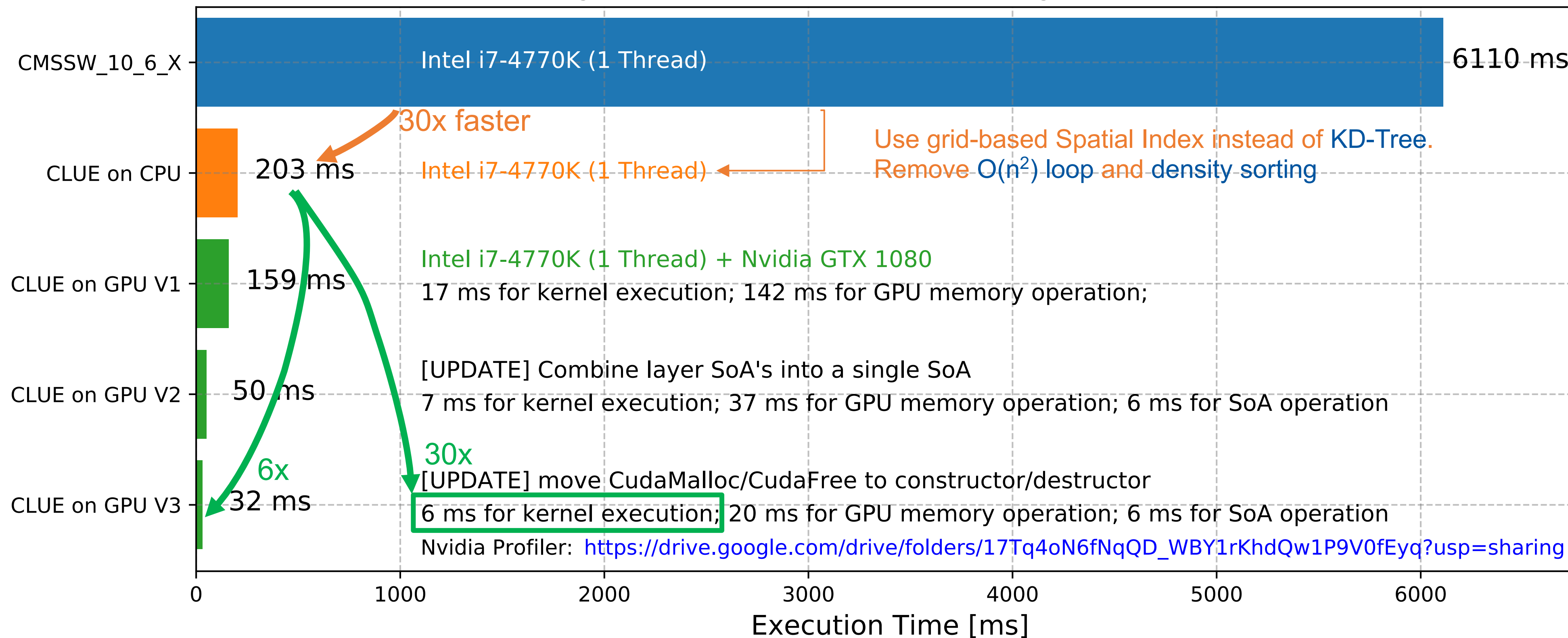


CLUE in CMSSW framework running on CPU and GPU

- CLUE has almost (>99%) identical clustering result as previous clustering algorithm in CMSSW. (very rare cases having difference are studied and show little impact)
- The previous clustering used KD-Tree for neighborhood query and had an $O(n^2)$ loops and $O(n\log(n))$ density sorting. [1]

[1] previous algorithm is described here but core/halo step is removed
http://hgcal.web.cern.ch/hgcal/Reconstruction/imagingAlgorithm/

CLUE in CMS Software (CMSSW) Framework



- CLUE CPU has about 30X speed up over previous clustering in CMSSW.
- CLUE GPU V3 gives an additional 6X on top of CLUE CPU in CMSSW framework.
- GPU run time (32 ms) also includes memcpy between host and device (20 ms) and SoA conversion (6 ms). But they can be shared by other GPU reconstruction steps and can be partially hidden if multiple CUDA stream work on different events. 30X speed up over CPU if excluding memcpy and SoA conversion.

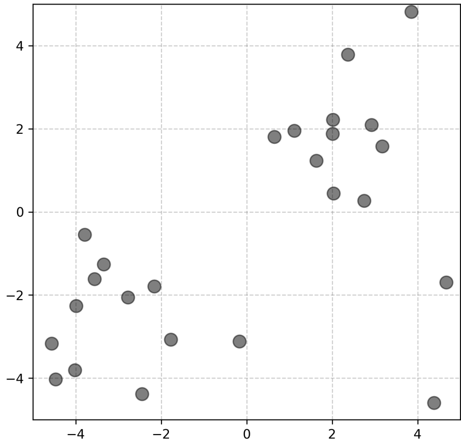
Conclusion

- We introduce CLUE, an O(n) complexity and GPU-friendly clustering algorithm.
- CLUE is ideal for heavy clustering task in CMS HGCAL reconstruction during HL-LHC.
- Thanks to CMS's plan of heterogeneous architecture in HLT and offline reconstruction, HGCal clustering can run on GPU and can provide promising acceleration.
- In CMS Software (CMSSW) framework, CLUE CPU is 30x faster (203 ms) than previous CPU clustering (6110 ms). CLUE on GPU gives another extra 6x speed up over CLUE on CPU (30x or 6 ms if excluding time of data traffic and SoA conversion).
  - penalty due to data traffic and SoA conversion will be shared with other GPU reconstruction steps.
  - these penalties can also be partially hidden if multiple CUDA streams works on different events.

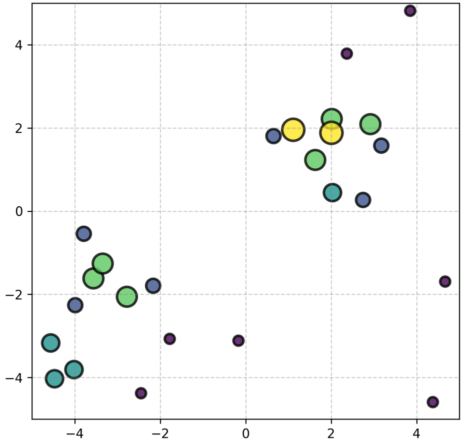
Backup

CLUE Procedure

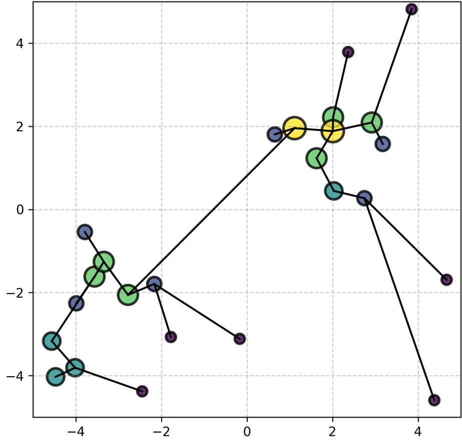
build data structure



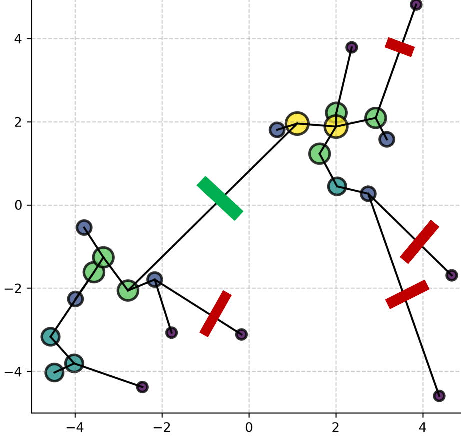
density



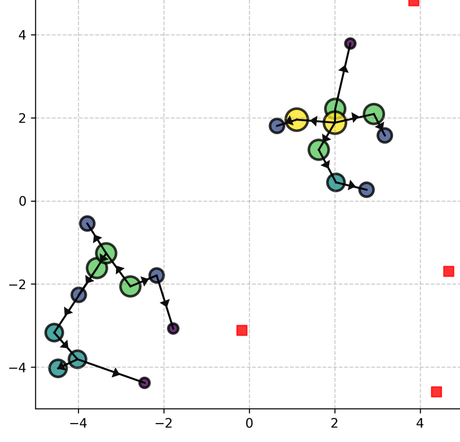
nearest higher



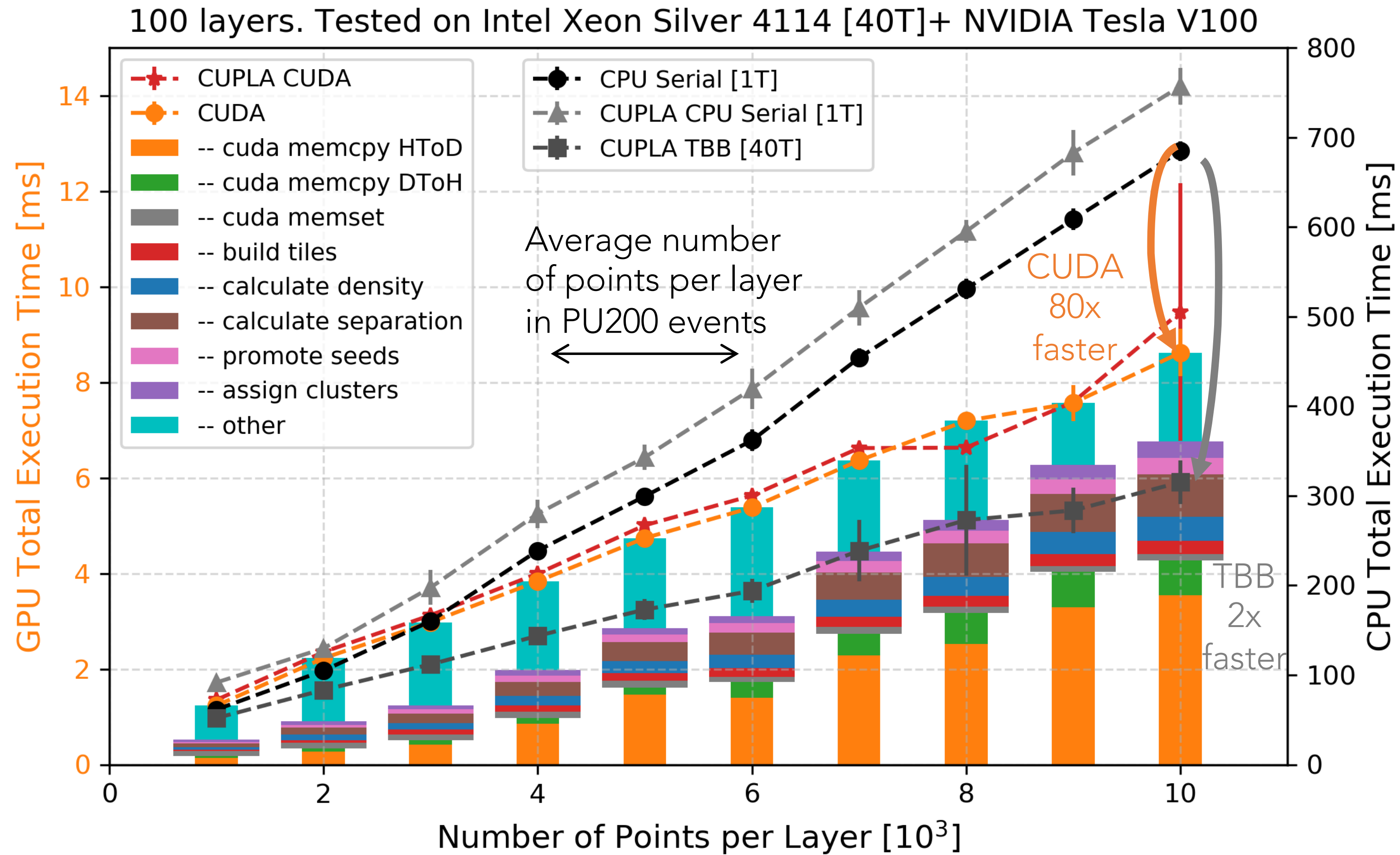
find seed



assign clusters



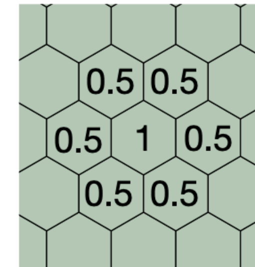
Clustering by Energy (CLUE) on GPU



Energy Density in the CMS HGCAL Clustering

Definition of energy density $$\rho_i = \sum_{j:j \in N_{d_c}(i)} \chi(d_{ij}) w_j$$

Density kernel in HGCAL



For CMS HGCAL, w_j is energy of hit and $d_c = 13$ mm, which equals to the distance between two adjacent cells. The density kernel used in HGCAL is

$$\chi(d_{ij}) = \begin{cases} 1 \; if \; i = j \\ 0.5 \; if \; 0 < d_{ij} < d_c \end{cases}$$

Comparing with a single cell of maximum local energy, local energy density is more sensitive to a multi-cell blob of energy.