GPU-based algorithms for the CMS track clustering and primary vertex reconstruction

Connecting The Dots 2022,

June 2nd 2022, Princeton

Carlos Erice on behalf of

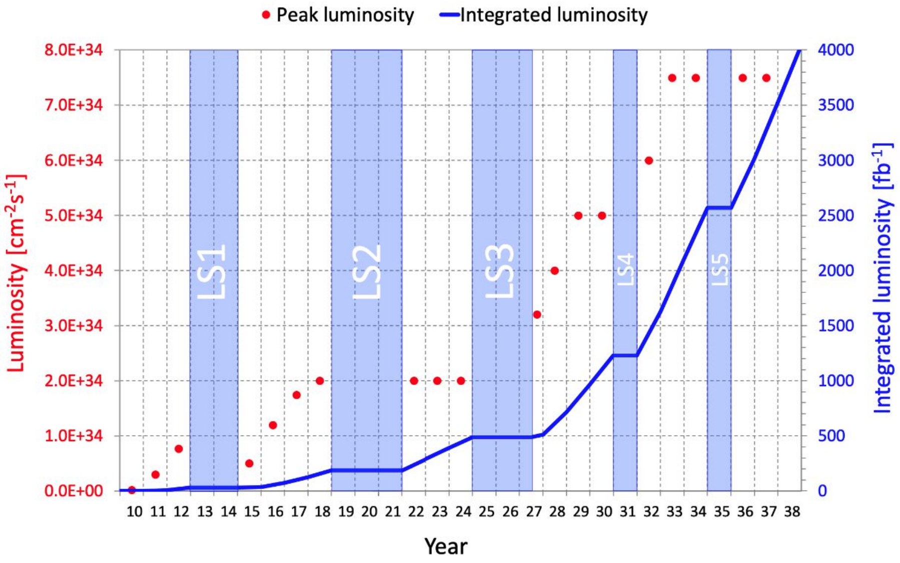
the CMS Collaboration

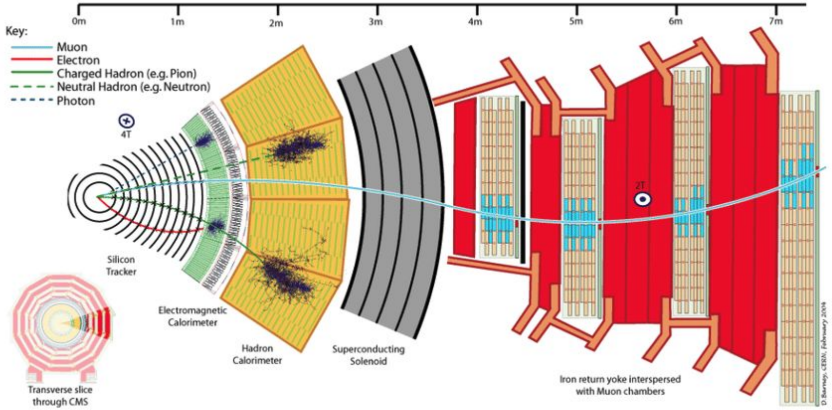
CMS - from Run 2 to beyond

Several changes incoming from the Run 3 and Phase II of the LHC for both online (trigger) and offline reconstruction:

→ Significant instantaneous luminosity increase: more data taken per second. Need a triggering system with a fast an efficient response to guarantee physics coverage (see Adriano's talk). The complexity of some reconstruction algorithms will also increase.

→ But also integrated luminosity: more data taken overall. Reconstruction algorithms used to obtain a faithful offline physics reconstruction will need to run over significantly more data.



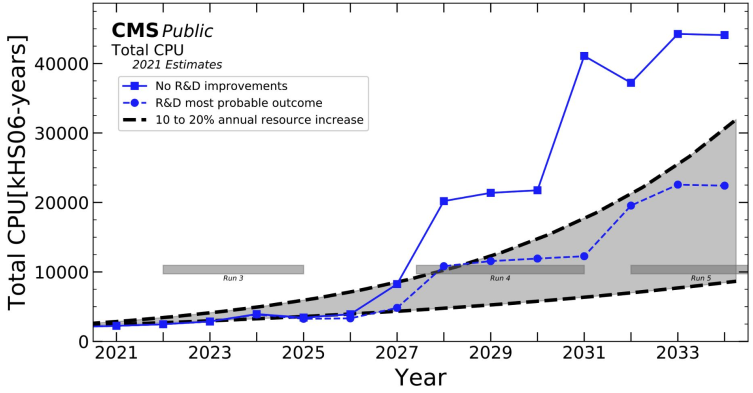


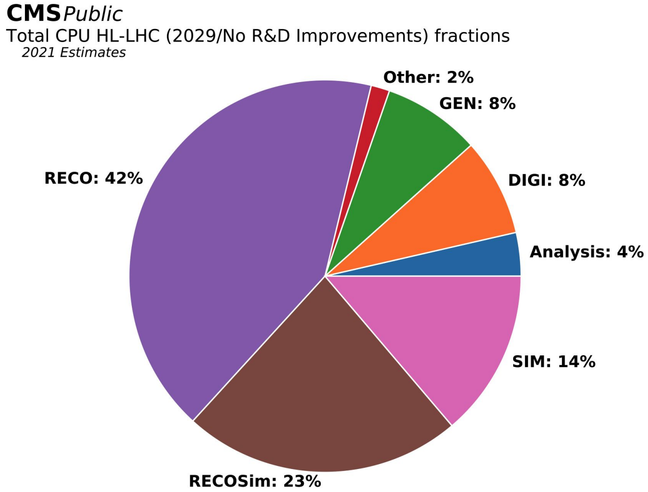
Computational challenges

→ With the current computing capabilities CMS would quickly run out of computational power to address the needs of the Phase II of the LHC: new R&D is needed to reduce our CPU needs.

→ Roughly ¾ of the pie already taken by reconstruction-related tasks.

→ The successful offloading of CMS HLT tracking/vertexing tasks to GPU-based architectures shows an open path for offline reconstruction to follow.





Estimations for:

(Run 4) 270 fb^{-1}/year at <PU>= 140
(Run 5) 350 fb^{-1}/year at <PU>= 200

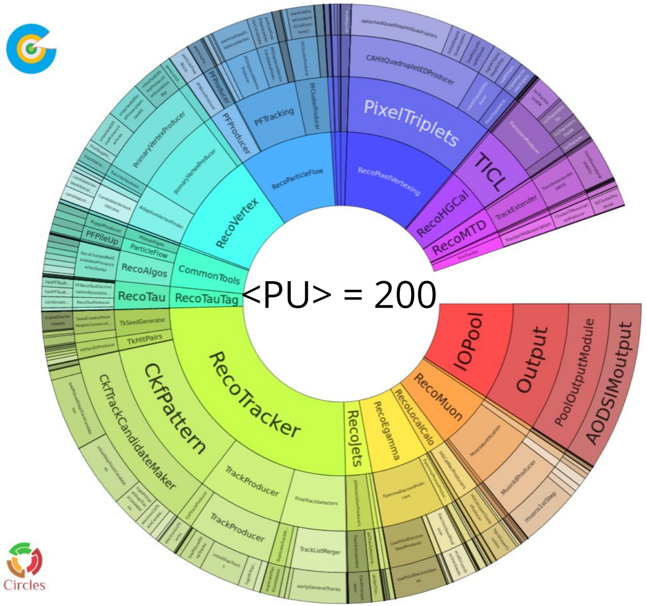
Vertexing at CMS - A global picture

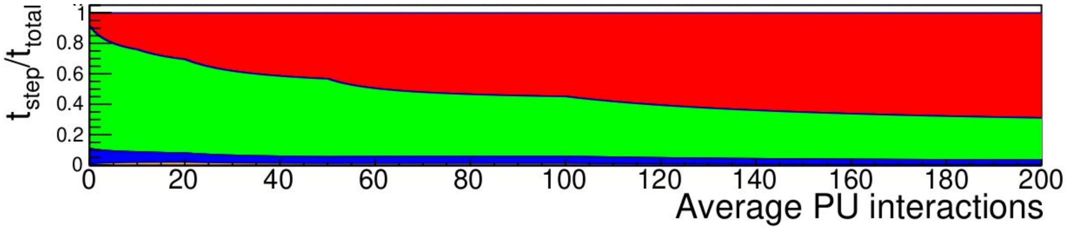
→ Roughly 10% of the overall time of the reconstruction time, CMS primary vertex (PV) reconstruction proceeds in roughly 3 steps:

→ Track selection: preparation of inputs.

→ Clustering: grouping of nearby tracks into sets of vertex candidates.

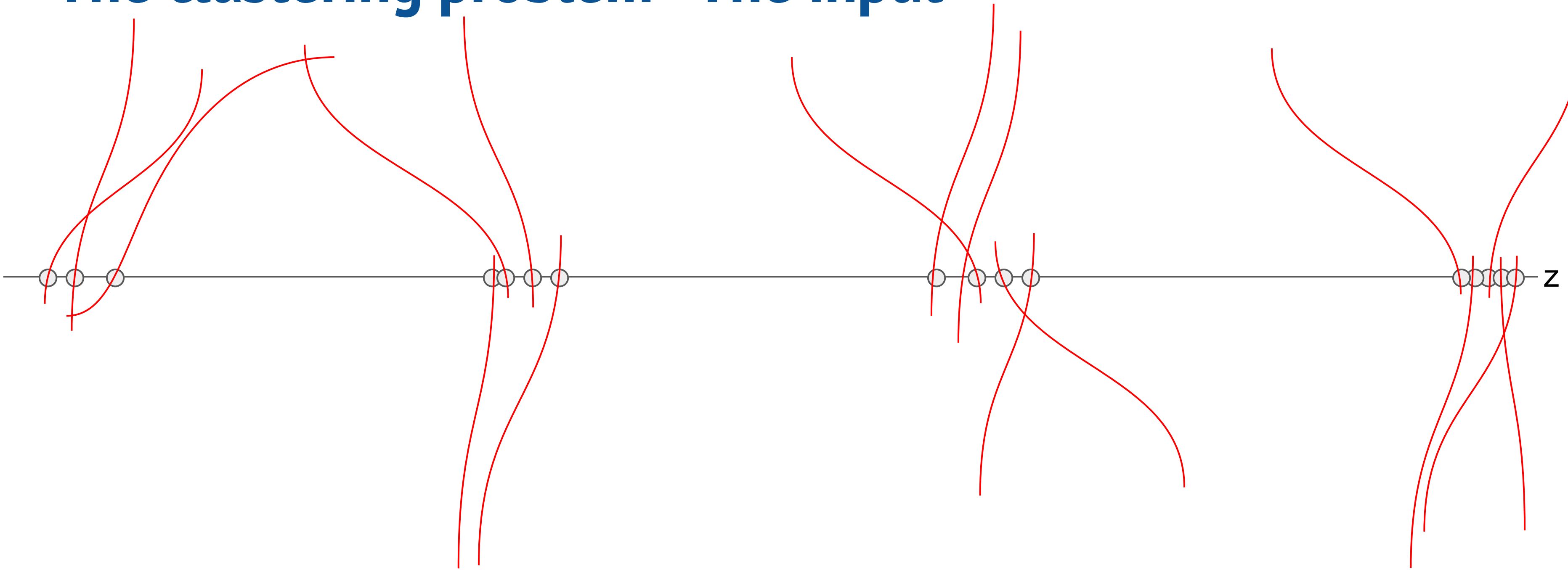
→ Fitting: obtain properties of the vertex themselves.





→ Towards higher <PU> values, the most HL-LHC like regions, clustering tracks takes most of the running time.

The clustering problem - The input



→ For PV reconstruction, all reconstructed tracks in the detector are filtered based on a small set of quality criteria: based on the amount of pixel/tracker hits, consistency in the transverse plane, etc.

→ Then they are reduced to **their positions -*z_i*-** at the point of closest approach to the beam and the **uncertainty of these positions -*$\sigma(z_i)$*-**.

The clustering process - Hard clustering

→ "Hard clustering": assign univocally each track to a given vertex candidate (cluster). It requires finding several quantities: number of vertex "K", their positions "z_k", and the track-vertex assignment $P_{ik} = 0 \text{ or } 1$

$$E = \sum_{i=1}^{I} \sum_{k=1}^{K} P_{ik} \frac{(z_i - z_k)^2}{\sigma_i^2}$$

→ Can roughly be presented as a direct optimization problem given the set of input tracks and their properties. Plenty of available solutions for a direct optimization problem.

Issues:

→ The non-fixed nature of "K", means that the number of vertex candidates needs to be extracted from data.

→ Any clustering needs to avoid local minima, stable solutions far from optimal settings (i.e. one cluster candidate per input track would be optimal but very much unphysical).

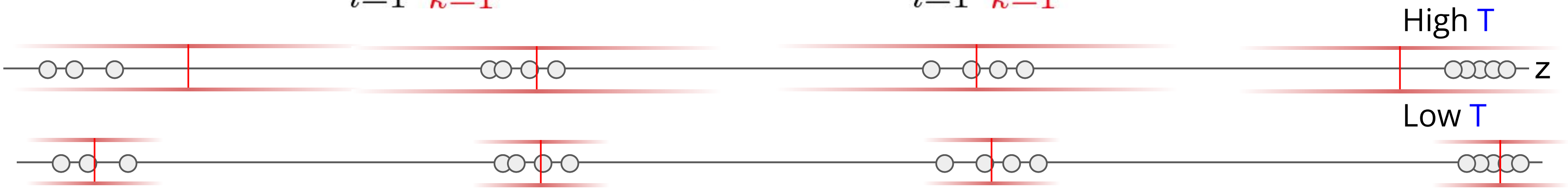
Deterministic annealing - I

→ A known robust solution is deterministic annealing (i.e. doi.org/10.1016/0031-3203(91)90097-O):

→ Hard assignment becomes a fuzzy one, P_{ik} != 0 or 1

→ An additional term is added to the minimization, acting as an entropy regulated by a temperature T:

$$E - TS = \sum_{i=1}^{I}\sum_{k=1}^{K} P_{ik}\frac{(z_i - z_k)^2}{\sigma_i^2} + T\sum_{i=1}^{I}\sum_{k=1}^{K} P_{ik}\log(P_{ik})$$

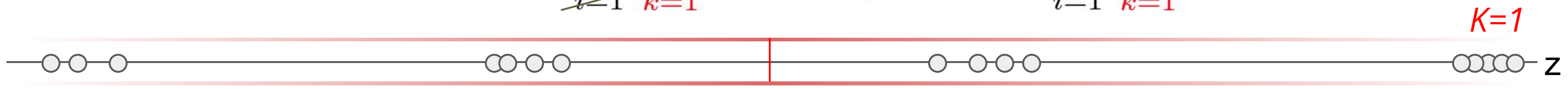


→ At high T assignment becomes fuzzier => overall broader clusters, at low T it is the opposite.

Deterministic annealing - II

→ The core of the algorithm relies on a start at very high T, for which a single vertex with all tracks is an analytical solution

$$E - TS = \cancel{\sum_{i=1}^{I}\sum_{k=1}^{K} P_{ik}\frac{(z_i - z_k)^2}{\sigma_i^2}} + T\sum_{i=1}^{I}\sum_{k=1}^{K} P_{ik}\log(P_{ik})$$



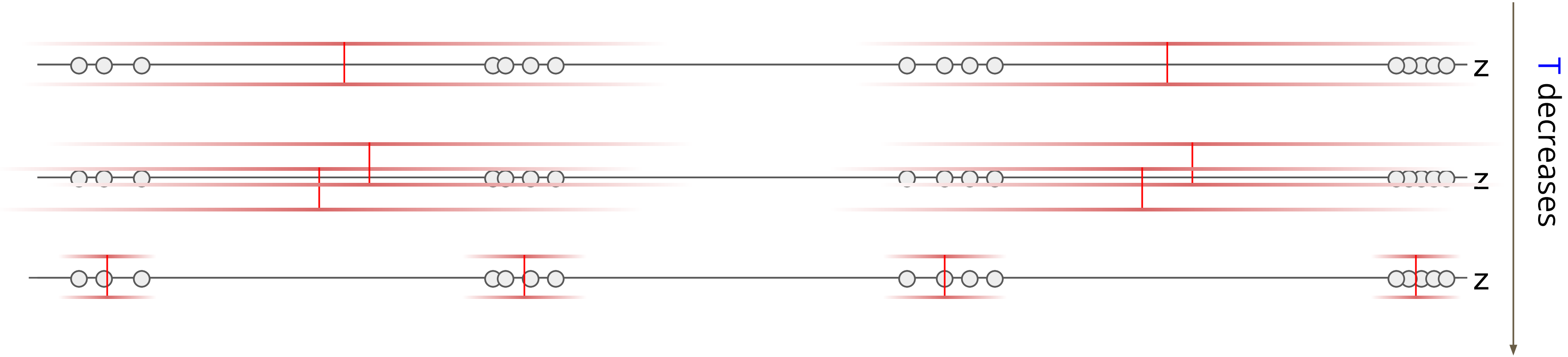
→ Slowly decreasing T, a critical temperature is reached where it is better to split the vertex in two:

→ The assignment probabilities and vertex positions can be solved by minimizing “E-TS” at fixed T:

$$\left.\begin{matrix} P_{ik} = P_{ik}(z_i, z_k; T) \quad z_k = z_k(z_i, P_{ik}; T) \end{matrix}\right\}$$ Applied iteratively with **<u>small steps in T</u>**

T decreases

Deterministic annealing - III

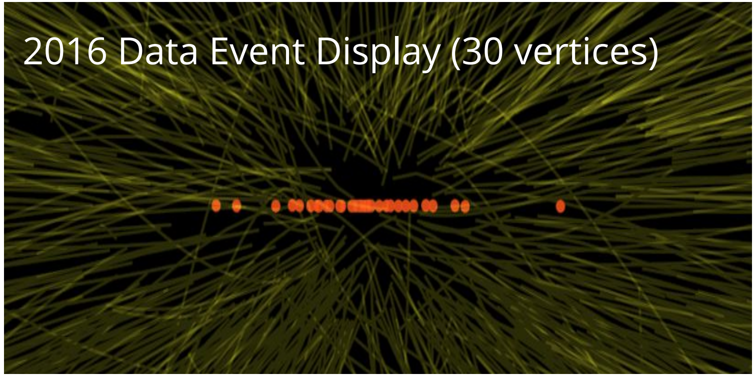


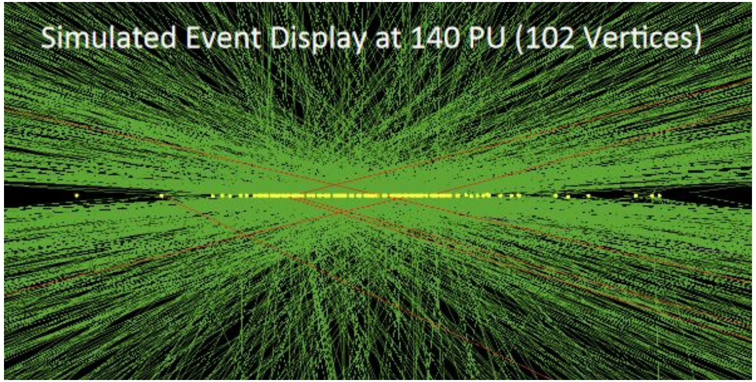
→ The algorithm continues alternating between splitting vertices and decreasing T until a minimal temperature (~minimum cluster size) is reached to define the final set of track clusters or "vertices".

→ Main advantages: convergence to the globally optimal solution, clear definition of the number of final clusters through the overall algorithm.

→ Main disadvantages: complexity of the algorithm, need to do ~#tracks x #vertices operations per T loop which might introduces a significant number of computations.

GPU implementation - Advantages and challenges





The algorithm complexity increases significantly as <PU> increases:

→ "Run II": ~20 vertex, ~ 500 tracks => 10^4 P_{ik} parameters

→ "Phase II": ~200 vertex, ~5000 tracks => 10^6 P_{ik} parameters.

As the operations to update P_{ik}/z_k are done iteratively, each step in T is, a priori, a great candidate for improvement through the usage of GPU architectures. Several challenges appear, though:

→ Need to keep track of ~10^6 parameters across T iterations.

→ Organize the code for efficiently parallelize operations without the appearance of running conditions -i.e. need to synchronize all threads after each T iteration-.

GPU implementation - storage challenges

→ Main challenge is the non-trivial organization+storage of data:

→ P_{ik} stored as a SoA matrix included into the track objects.

→ z_k a simple array of positions.

→Operations can then be summarized as per-matrix entry operations and sums “across rows”.

Tracks:

$t1: z_1, dz_1, \ldots, (P_{11}, \ldots, P_{1K}, 0, \ldots, 0)$

$t2: z_2, dz_2, \ldots, (P_{21}, \ldots, P_{2K}, 0, \ldots, 0)$

.

.

.

$tN: z_1, dz_1, \ldots, (P_{N1}, \ldots, P_{NK}, 0, \ldots, 0)$

Vertices:

$v_1: z_1$

$v_2: z_2$

.

$v_K: z_K$

$v_{K+1}: 0$

.

$v_{512}: 0$

→ Rather unfortunately, as the overall number of vertices is a priori unknown, an oversized space in the GPU memory is reserved -and zero padded- before the algorithm is run. Amount of allocated memory set to catch possible high PU (up to 300 vertex) events in Phase II conditions.

GPU implementation - multithreading challenges

→ Bulk of operations ~10^6 exponentials readily parallelizable across all entries:

(Actual formulas slightly more complex)

$$p_{ij}^{(n)} = \frac{e^{-\beta\left(\frac{z_i - z_k^{(n-1)}}{\sigma_i^2}\right)^2}}{\sum_{l=1}^{K} e^{-\beta\left(\frac{z_i - z_l^{(n-1)}}{\sigma_i^2}\right)^2}}$$

Step in T loop

→ Vertex positions are parallelized across all vertices as well:

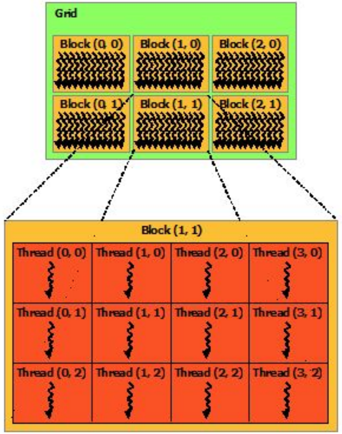
$$z_k^{(n)} = \frac{\sum_{i=1}^{I} P_{ik}^{(n-1)} z_i}{\sum_{i=1}^{I} P_{ik}^{(n-1)}}$$

Tracks:

t1: z_1, dz_1, …, (P_{11}, ..., P_{1K},0,...,0)
t2: z_2, dz_2, …, (P_{21}, ..., P_{2K},0,...,0)
.
.
.
tN: z_1, dz_1, …, (P_{N1}, ..., P_{NK},0,...,0)

Vertices:

v_1: z_1
v_2: z_2
.
v_K: z_K
v_{K+1}: 0
.
v_{512}: 0



Ultimately limited by two factors:

- Amount of threads that can be run in parallel.

- Need to synchronize operations in each step on T (i.e., wait for slowest thread to finish computations).

GPU implementation - performance and plans

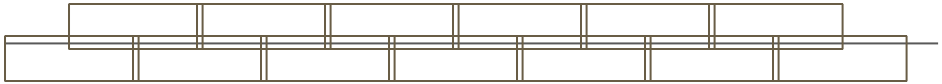
→ A first implementation has been done to run on CUDA-based GPU systems.

→ Full synchronization between CPU and GPU implementations of the deterministic annealing algorithm. Thus, the performance of the current CMS algorithm would be ensured in the CPU version.

→ Preliminary timing comparisons show comparable timings (~1s/event) between first CPU tests (IBRS Broadwell with a 2.2 GHz clock) and GPU ones (Tesla T4 with a 1.6GHz GPU clock).

→ Still some possibilities to explore for the future:

Clustering by blocks?



→ Reduce the amount of T steps/overall computations by pre-splitting tracks in small regions of space => Reduce the problem to several Run II-like iterations of the clustering that can be run in parallel!

Summary

→ Discussed a first implementation of offline Primary Vertexing for the CMS experiment in GPU-based architectures.

→ One of the steps needed to reduce the overall computational needs of the experiment.

→ A first implementation shows promising results in comparison with a CPU one, fully reproducing the current performance of CMS' primary vertex algorithms and showing similar results in terms of timing.

→ As more items of the reconstruction chain are offloaded to GPUs, significant improvements can be achieved in the overall CMS computing budget.

Backup