

# Seeding with a flat(ter) EDM

[PR #326](#)



# Fundamentals

- Jagged vectors are complicated to handle
  - “Fill prefix sum” kernel for translating 1D to 2D coordinates
  - Increased host operations between kernels
- New seeding still has jagged structure in the binned spacepoints (2D detray grid)
- Jaggedness eliminated in:
  - Doublets
  - Triplets

# Fundamentals

- Seed filtering requires comparison between triplets sharing 1 or 2 spacepoints.
- In a flat EDM, need to know where all these triplets to compare are located
  - If we just iterate over all triplets instead, things get really slow.
- Counting + filling approach is quite powerful
- Allows pre-defining vector locations for where items should be filled

# Flat Seeding EDM

```

17 // Doublet of middle-bottom or middle-top spacepoints
18 struct device_doublet { ← Two vectors of device doublets (bottom & top doublets)
19     // bottom (or top) spacepoint location in internal spacepoint container
20     sp_location sp2;
21
22     using link_type = device::doublet_counter_collection_types::host::size_type;
23     // Link to doublet counter where the middle spacepoint is stored
24     link_type counter_link;
25 }; ← Information regarding middle sp stored via link
16 // Number of doublets for one specific middle spacepoint.
17 struct doublet_counter { ← A counter for each middle sp
18
19     // Index of the middle spacepoint.
20     sp_location m_spM;
21
22     // The number of compatible middle-bottom doublets
23     unsigned int m_nMidBot = 0;
24     // The number of compatible middle-top doublets
25     unsigned int m_nMidTop = 0;
26     // The position in which these middle-bottom doublets will be added
27     unsigned int m_posMidBot = 0;
28     // The position in which these middle-top doublets will be added
29     unsigned int m_posMidTop = 0;
30
31 // struct doublet counter


```

Number of bottom & top doublets & their atomically claimed positions

Positions reduce work in triplet finding

```
110 // Add the counts if compatible bottom *AND* top candidates were found for
111 // the middle spacepoint in question.
112 if ((n_mb_cand > 0) && (n_mt_cand > 0)) {
113
114     // Increment the summary values in the header object.
115     vecmem::device_atomic_ref<unsigned int> numMidBot(nMidBot);
116     const unsigned int posBot = numMidBot.fetch_add(n_mb_cand);
117     vecmem::device_atomic_ref<unsigned int> numMidTop(nMidTop);
118     const unsigned int posTop = numMidTop.fetch_add(n_mt_cand);
119     Claimed positions
120     // Add the number of candidates for the "current bin".
121     doublet_counters.push_back({spM, n_mb_cand, n_mt_cand, posBot, posTop});
122 }
```

Global counters  
incremented atomically

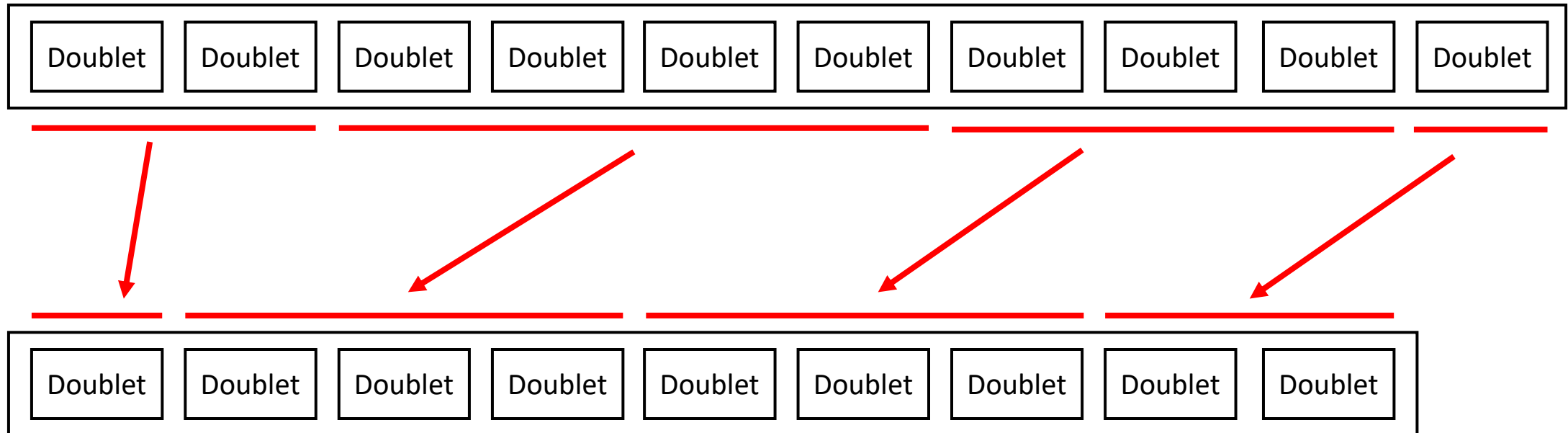


```
106 if (isCompatibleBot(middle_sp, other_sp)) {
107     const unsigned int pos = counter.mid_bot_start_idx + mid_bot_idx++;
108     mb_doublets.at(pos) = {middle_sp, other_sp};
109 }
```

Fill result in position obtained in counting



# Looking for triplets



- We now know where all the doublets which share the same middle spacepoint are
- This was already happening with the jagged EDM, but it gets a bit trickier from here on out

```
17  /// Triplets of bottom, middle and top spacepoints
18  struct device_triplet {
19      // top spacepoint location in internal spacepoint container
20      sp_location spT;
21
22      using link_type = device::triplet_counter_collection_types::host::size_type;
23      /// Link to triplet counter where the middle and bottom spacepoints are
24      /// stored
25      link_type counter_link; ← Information regarding middle & bottom sp stored via link
26
27      /// curvature of circle estimated from triplet
28      scalar curvature;
29      /// weight of triplet
30      scalar weight;
31      /// z origin of triplet
32      scalar z_vertex;
33  };
```



```

31 // Number of triplets for one specific Mid-Bottom Doublet.
32 struct triplet_counter { ← A counter for each mid-bottom doublet
33     // Bottom spacepoint location in internal spacepoint container
34     sp_location spB;
35
36     using link_type = triplet_counter_spM_collection_types::host::size_type;
37     // Link to the triplet counter per middle spacepoint
38     link_type spM_counter_link;
39     Information regarding middle sp stored via link
40     // The number of compatible triplets for this midbot doublet
41     unsigned int m_nTriplets = 0;
42     // The position in which these triplets will be added
43     unsigned int posTriplets = 0;
44 }; // struct triplet_counter

```

Number of triplets & their position

Allows comparison between these triplets in filtering

```

16 // Number of triplets for one specific middle spacepoint.
17 struct triplet_counter_spM { ← A counter for each middle sp
18     // Middle spacepoint location in internal spacepoint container
19     sp_location spM;
20
21     // The number of triplets for this middle spacepoint
22     unsigned int m_nTriplets = 0;
23     // The position in which these triplets will be added
24     unsigned int posTriplets = 0;
25 }; // struct triplet_counter_spM

```

Number of triplets & their position

Allows comparison between these triplets in filtering

```
105 // if the number of triplets per mb is larger than 0, write the triplet
106 // counter into the collection
107 if (num_triplets_per_mb > 0) {
108     triplet_counter_spM& header = spM_triplet_counter.at(counter_link);
109     vecmem::device_atomic_ref<unsigned int> nTriplets(header.m_nTriplets);
110     const unsigned int posTriplets =
111         nTriplets.fetch_add(num_triplets_per_mb);
112
113     mb_triplet_counter.push_back(
114         {spB_loc, counter_link, num_triplets_per_mb, posTriplets});
```

Triplet counters per spM incremented atomically

Claimed position

Collect spM counts to global counter

```
46 // Increment total number of triplets and claim position for this middle
47 // spacepoint's triplets
48 vecmem::device_atomic_ref<unsigned int> nTriplets(num_triplets);
49 this_spM_counter.posTriplets =
50     nTriplets.fetch_add(this_spM_counter.m_nTriplets);
```

Global triplet counter

Claimed position

# Reduced host operations

- With jagged structure we need to:
  - Copy the full vector of headers from the counting kernel
  - Allocate a jagged vector with the correct inner sizes
- With a flat structure:
  - Copy a single integer
  - Allocate a single large vector

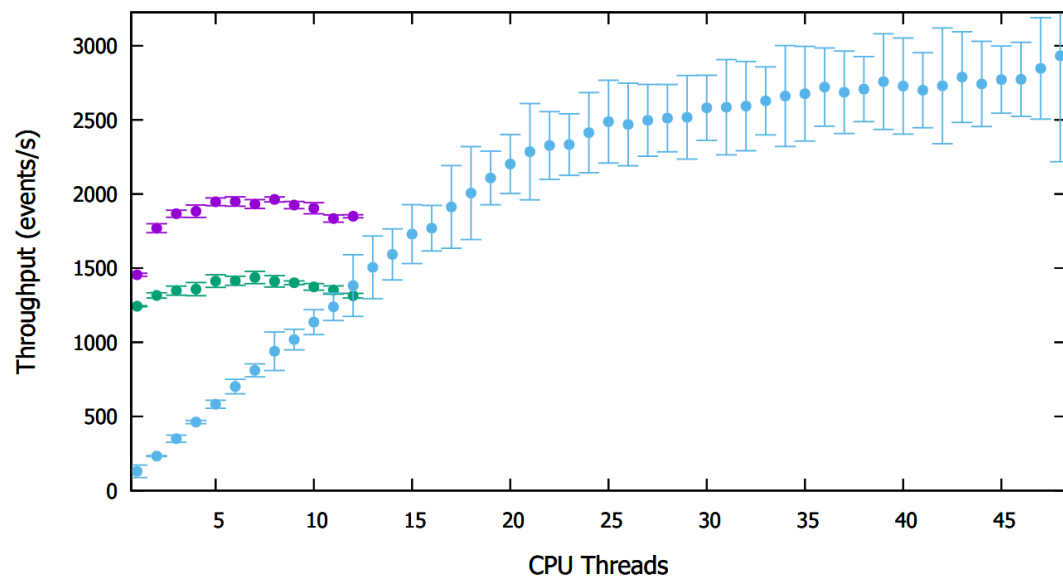
# Performance

- Full application from cells to bound tracks

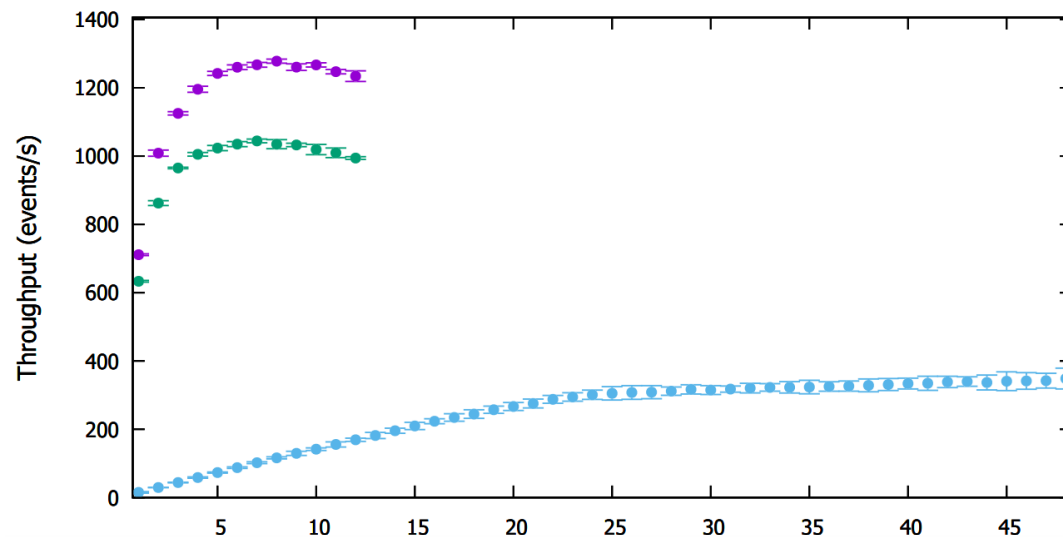
CPU: AMD EPYC 7413 (blue)

GPU: NVIDIA RTX A5000 (green – jagged ; purple – flat)

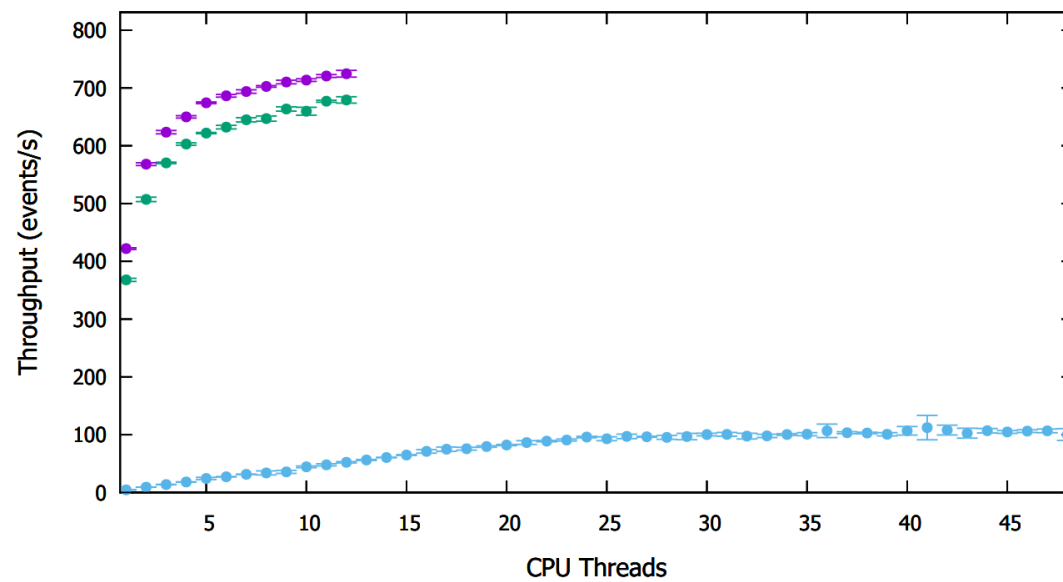
mu20



mu100



mu200



# Takeaways

- Separation between **internal** EDM used on device and CPU
- Linking between device doublets/triplets and their respective counters.
- Device doublets and triplets no longer binned by *sp\_grid*
  
- New seeding still has jagged structure in the binned spacepoints (2D detray grid)
- Jaggedness eliminated in:
  - Doublets
  - Triplets
- Removed some *fill\_prefix\_sum* kernels
- Added *reduce\_triplet\_counts* kernel
- Heavily simplified host operations between kernels