JÜLICH
FORSCHUNGSZENTRUM

panda

Update on GPU-based Online Tracking Algorithms

2014-4 Collaboration Meeting, Pattern Recognition

9 December 2014, Andreas Herten

Outline

- Updates on
  - Hough Transform
  - Triplet Finder

JÜLICH
FORSCHUNGSZENTRUM

panda

ALGORITHMS #1

Hough Transform

Triplet Finder

Hough Transform — Granularity

- Line Hough around point $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$

$\alpha = 0°, 2°, 4°, \ldots$

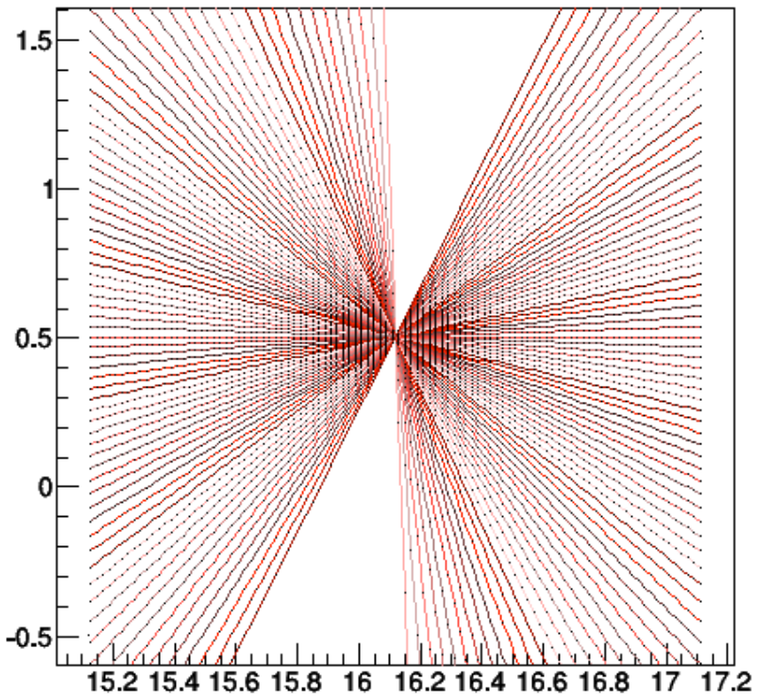
static slide

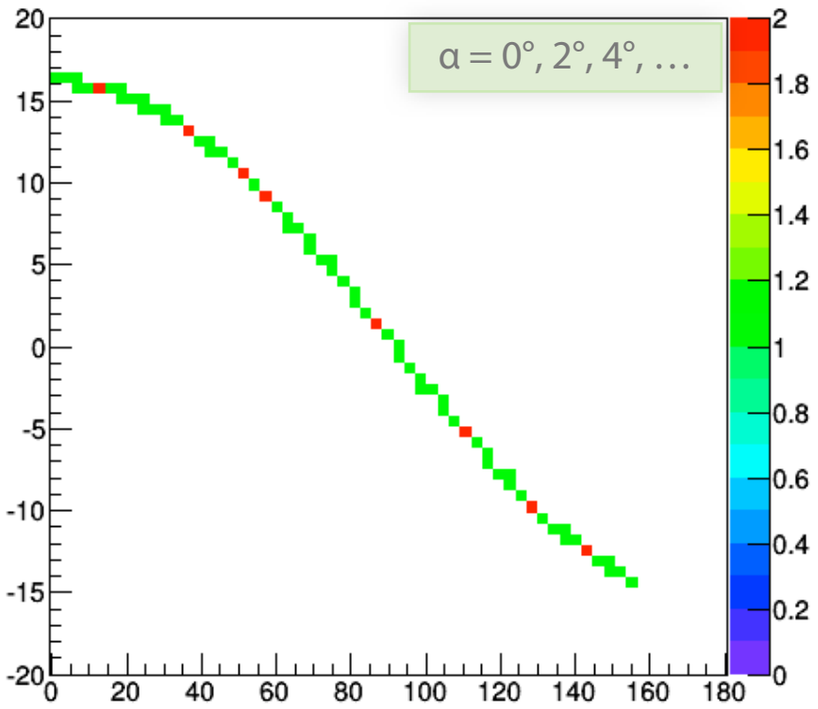
$\alpha = 0°, 2°, 4°, \ldots$

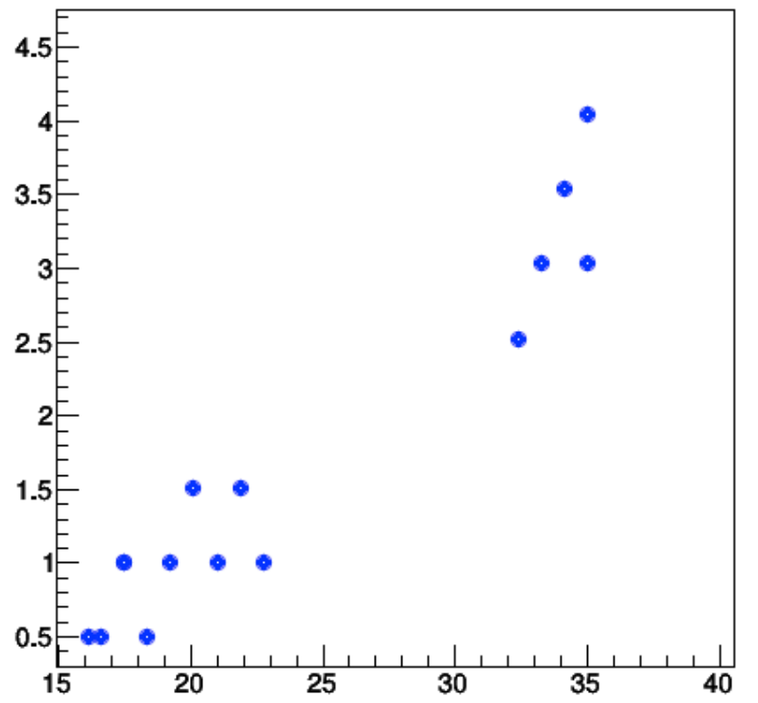
Hough Transform — Granularity

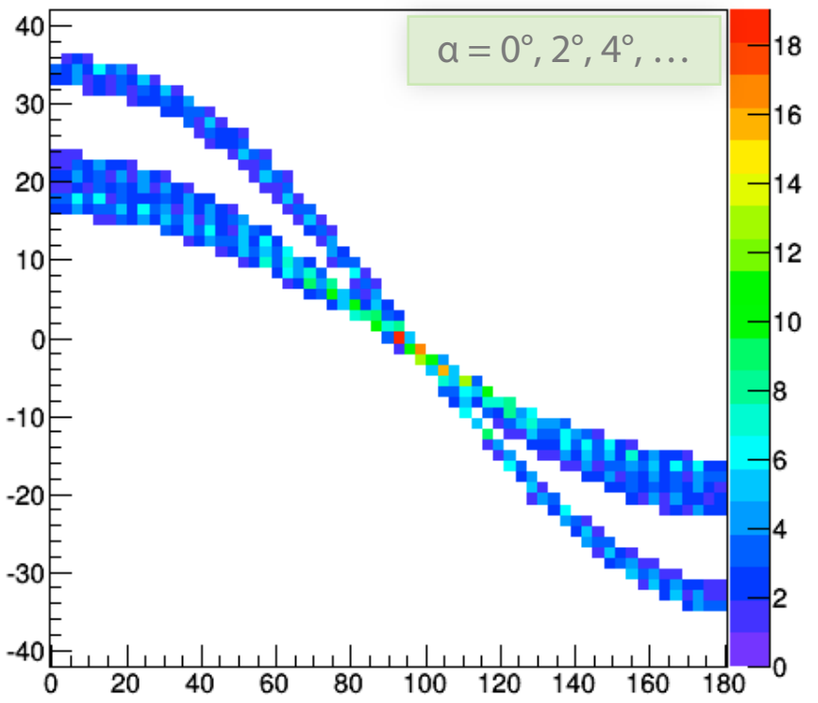
- Line Hough around point $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$

static slide





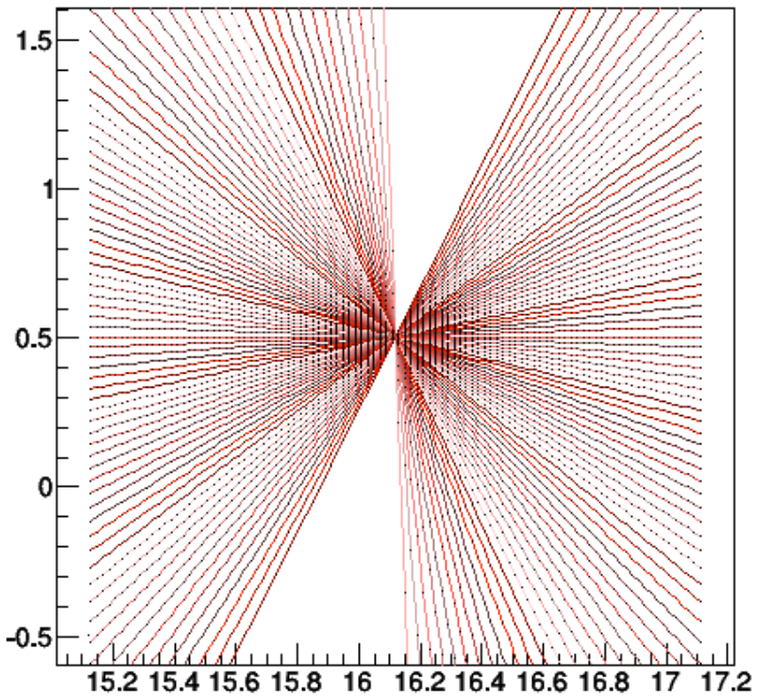


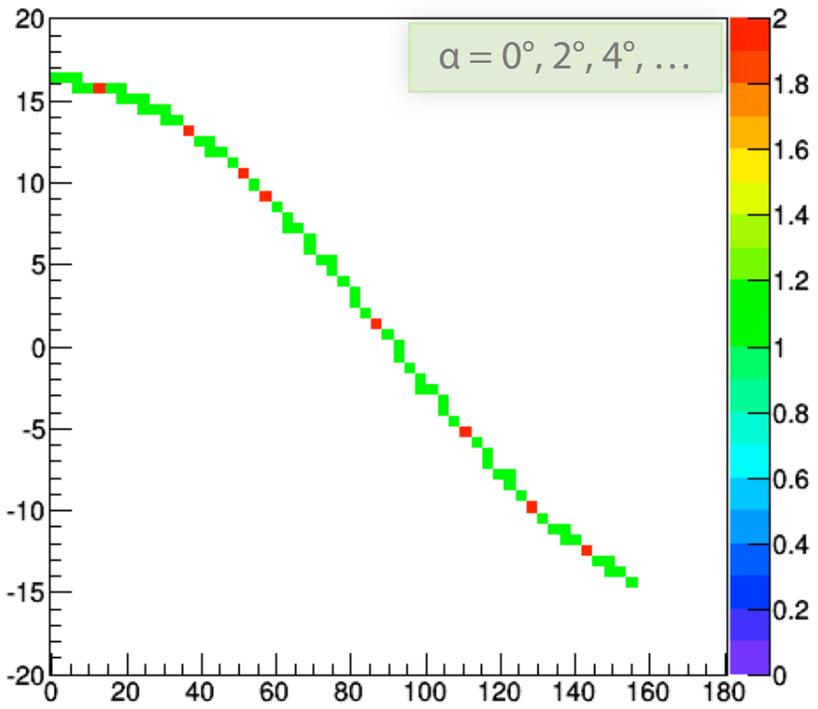


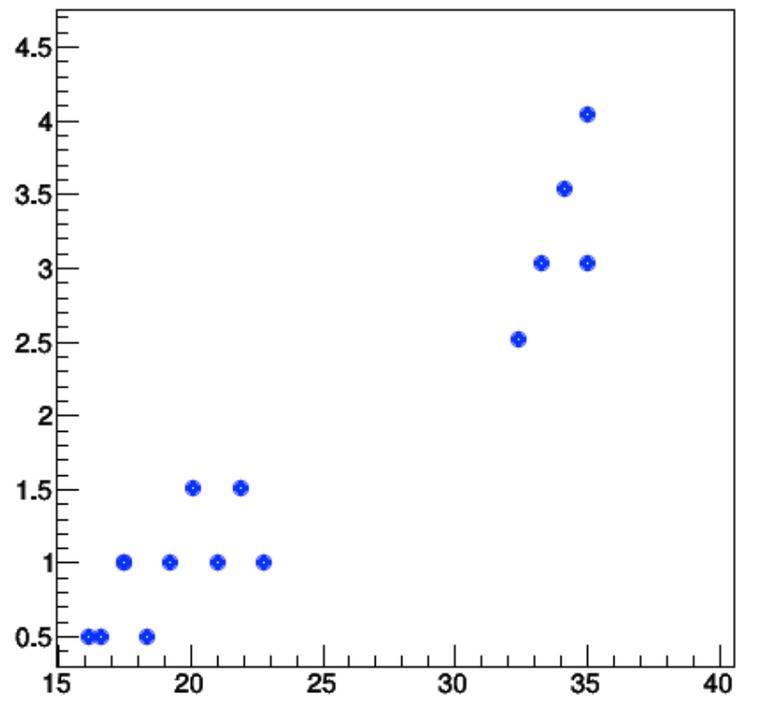
Hough Transform — Granularity

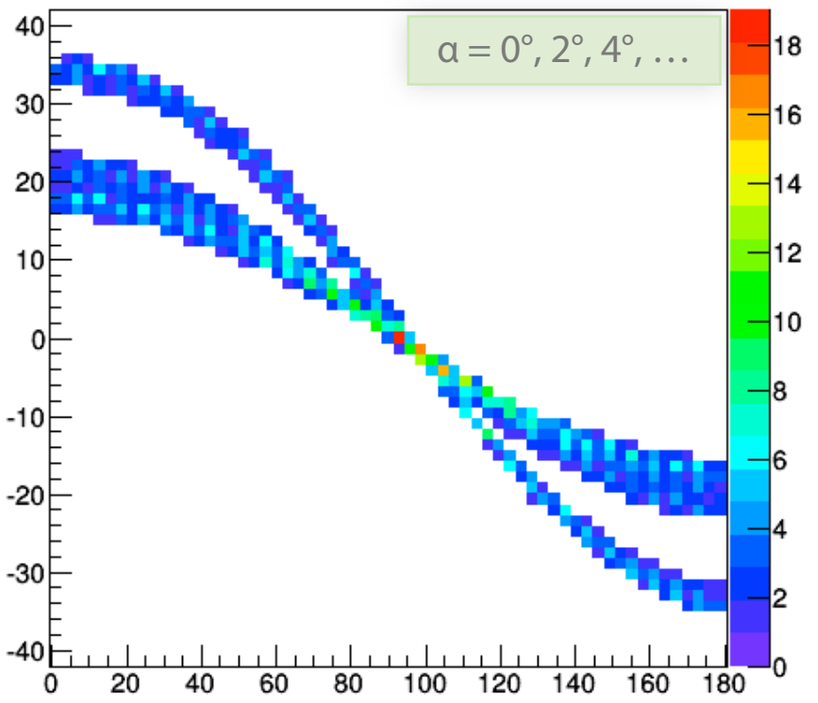
- Line Hough around point $r_{ij} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i$

static slide









Hough Transform — Variations

- Line Hough with isochrones $r_{ij}^{\pm} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i \pm \rho_i$

static slide

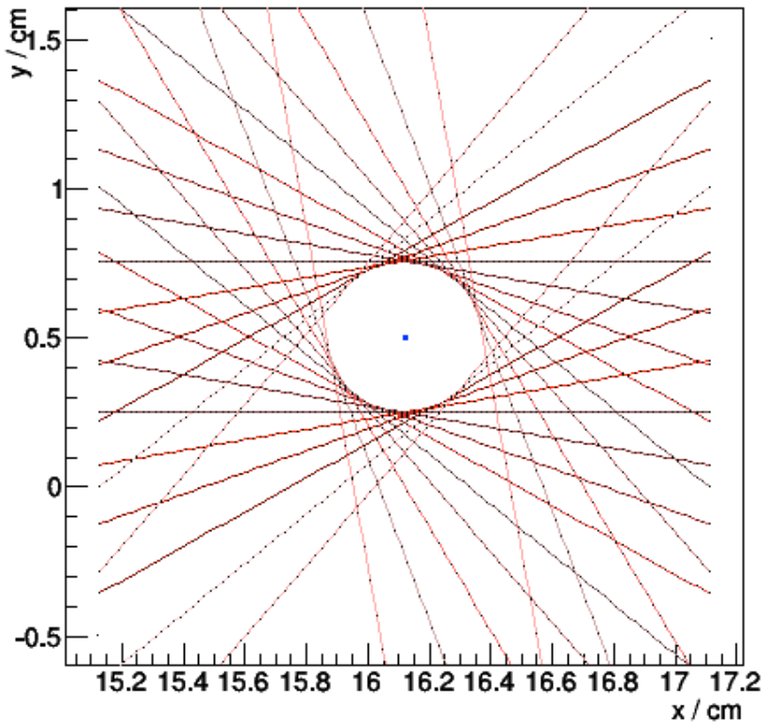
α = 0°, 10°, 20°, …

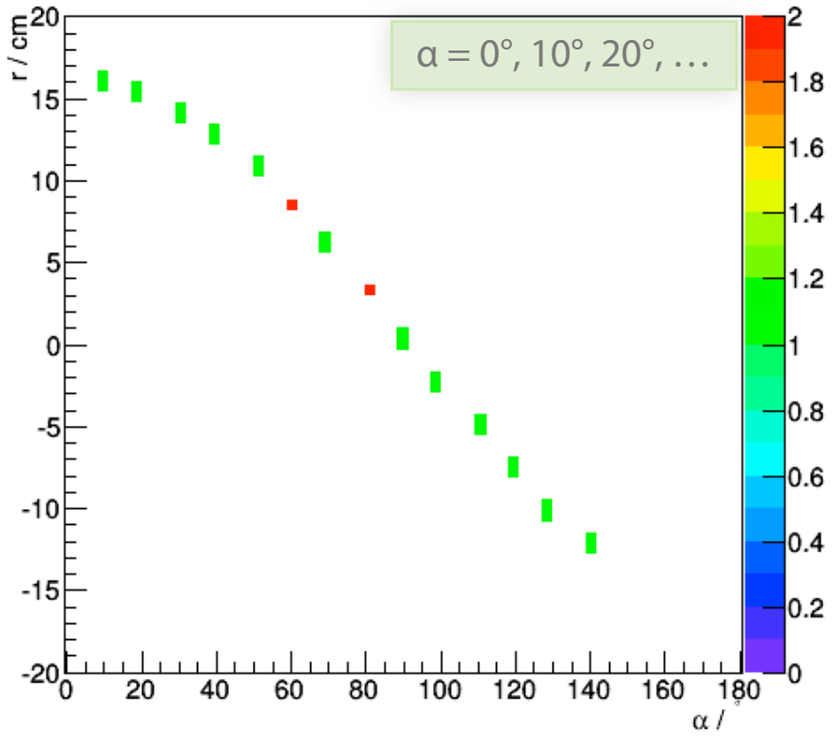
α = 0°, 10°, 20°, …

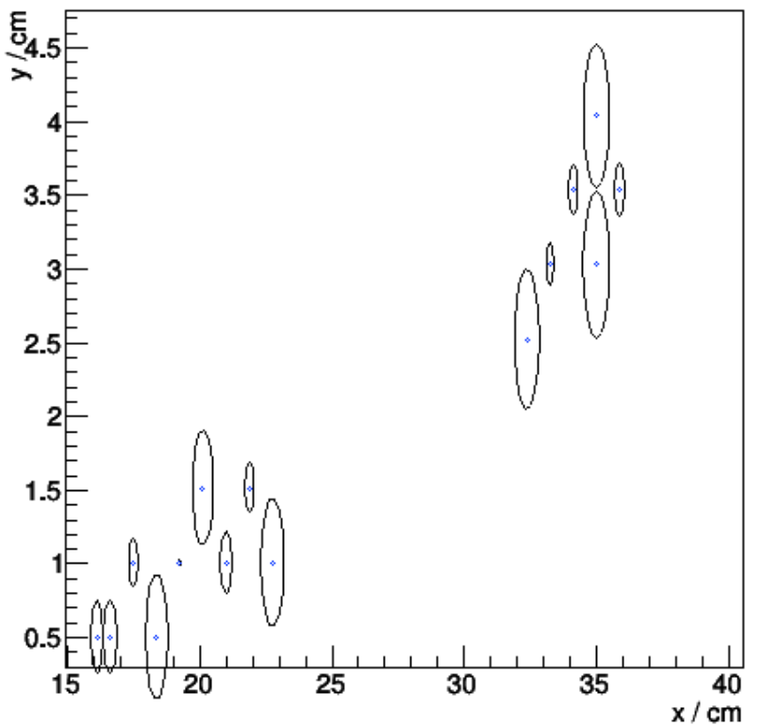
Hough Transform — Variations

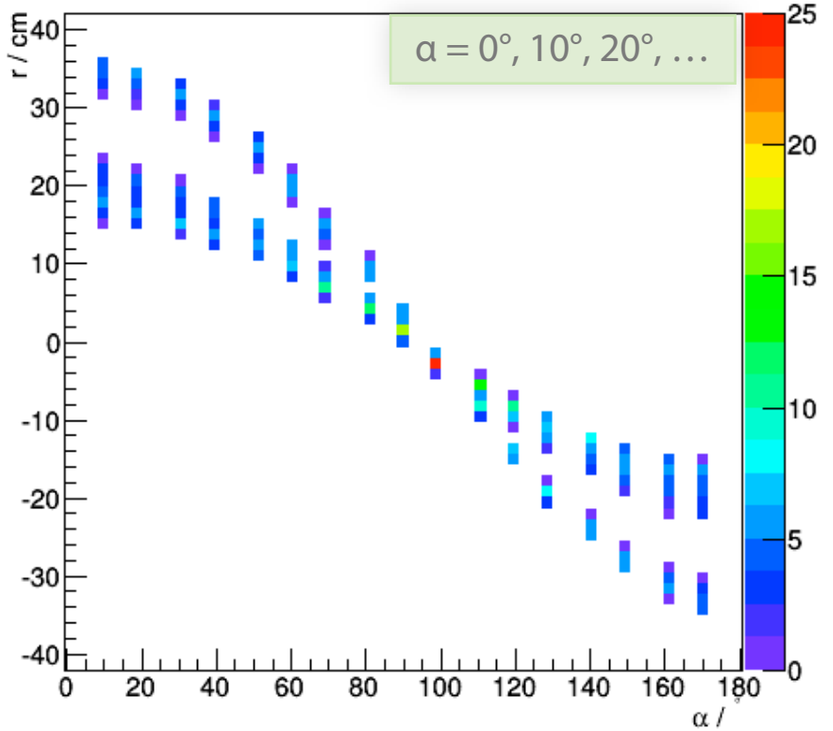
- Line Hough with isochrones $r_{ij}^{\pm} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i \pm \rho_i$

static slide





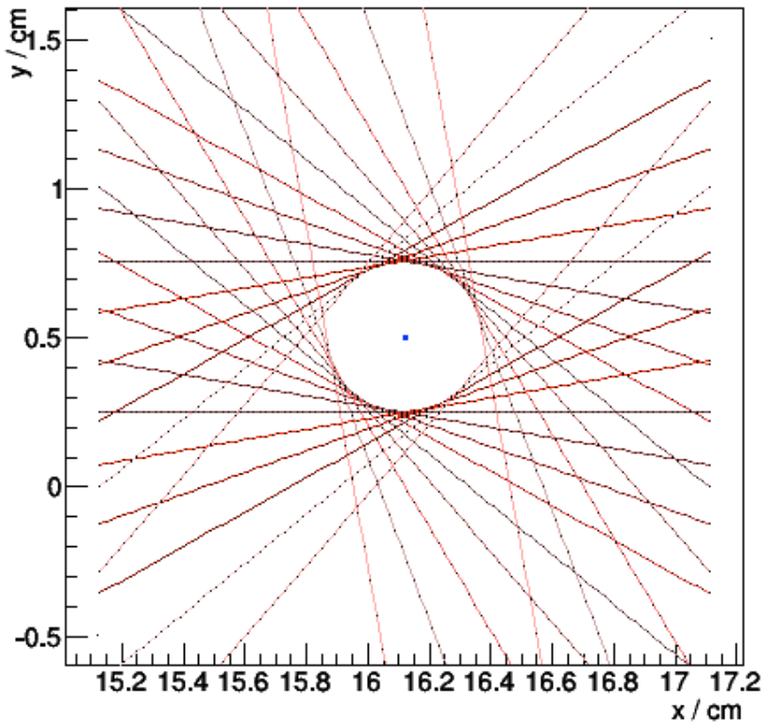


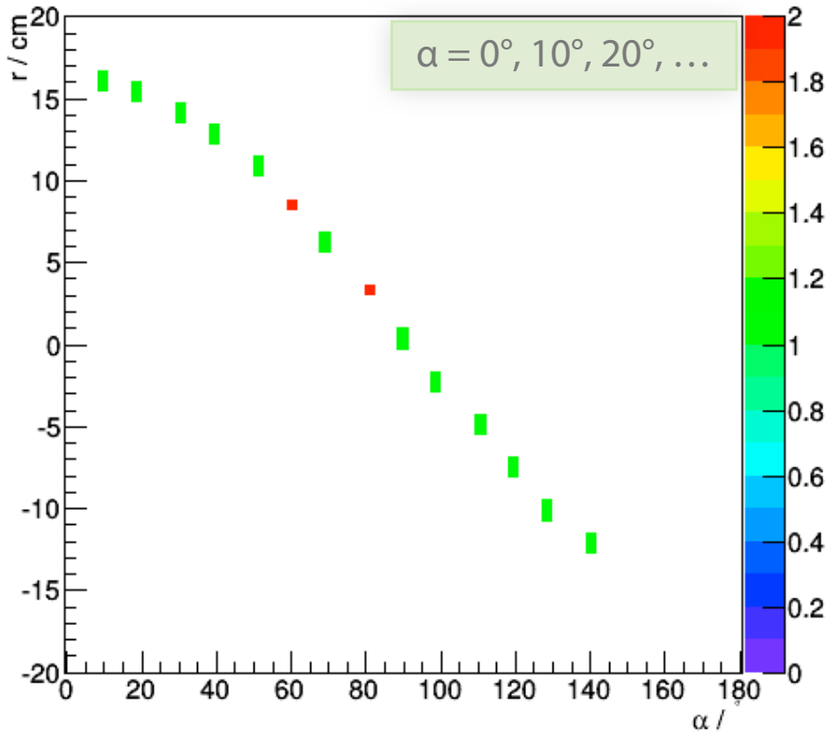


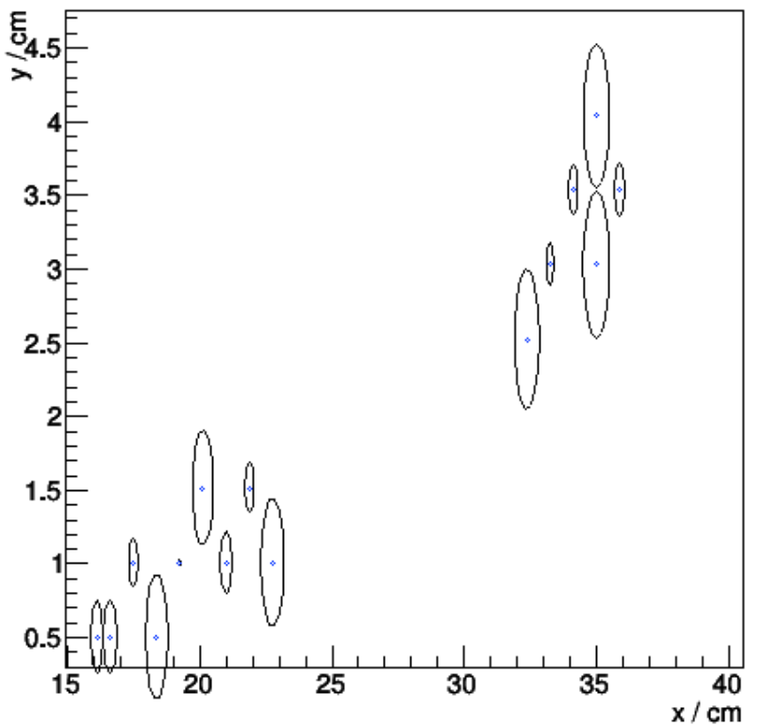
Hough Transform — Variations

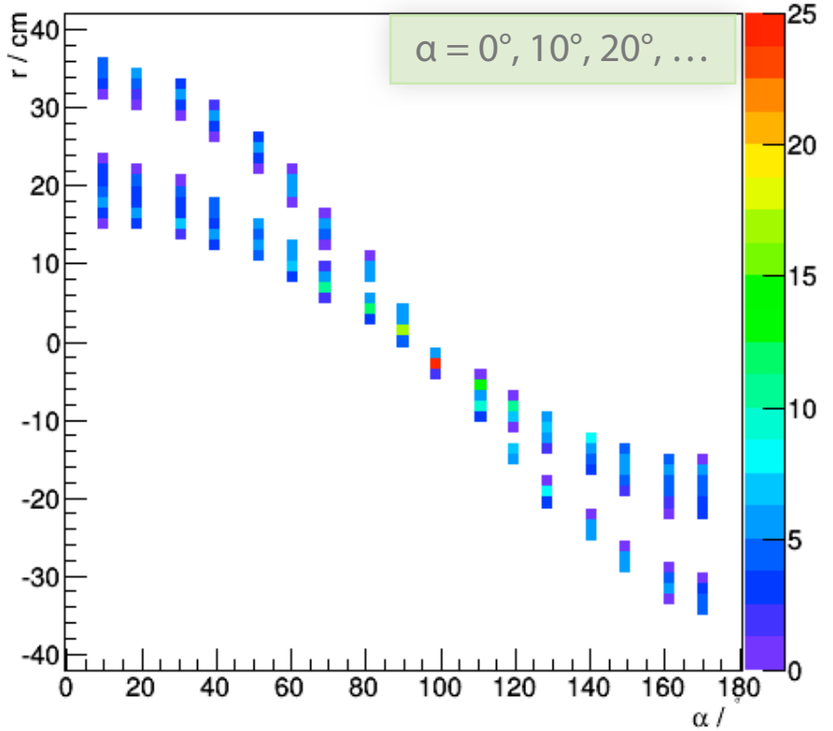
- Line Hough with isochrones $r_{ij}^{\pm} = \cos(\alpha_j) \cdot x_i + \sin(\alpha_j) \cdot y_i \pm \rho_i$

static slide

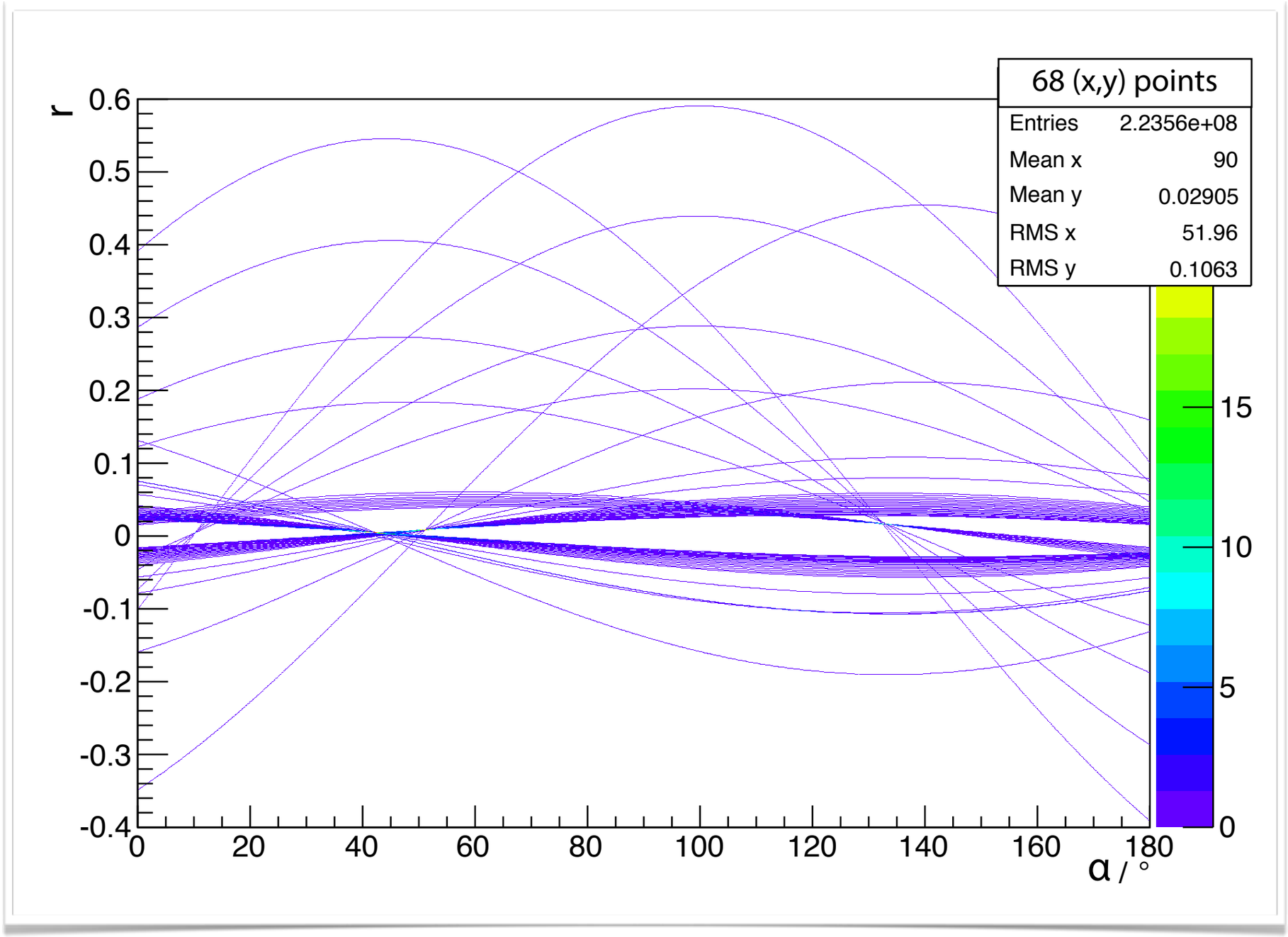








Hough Transform



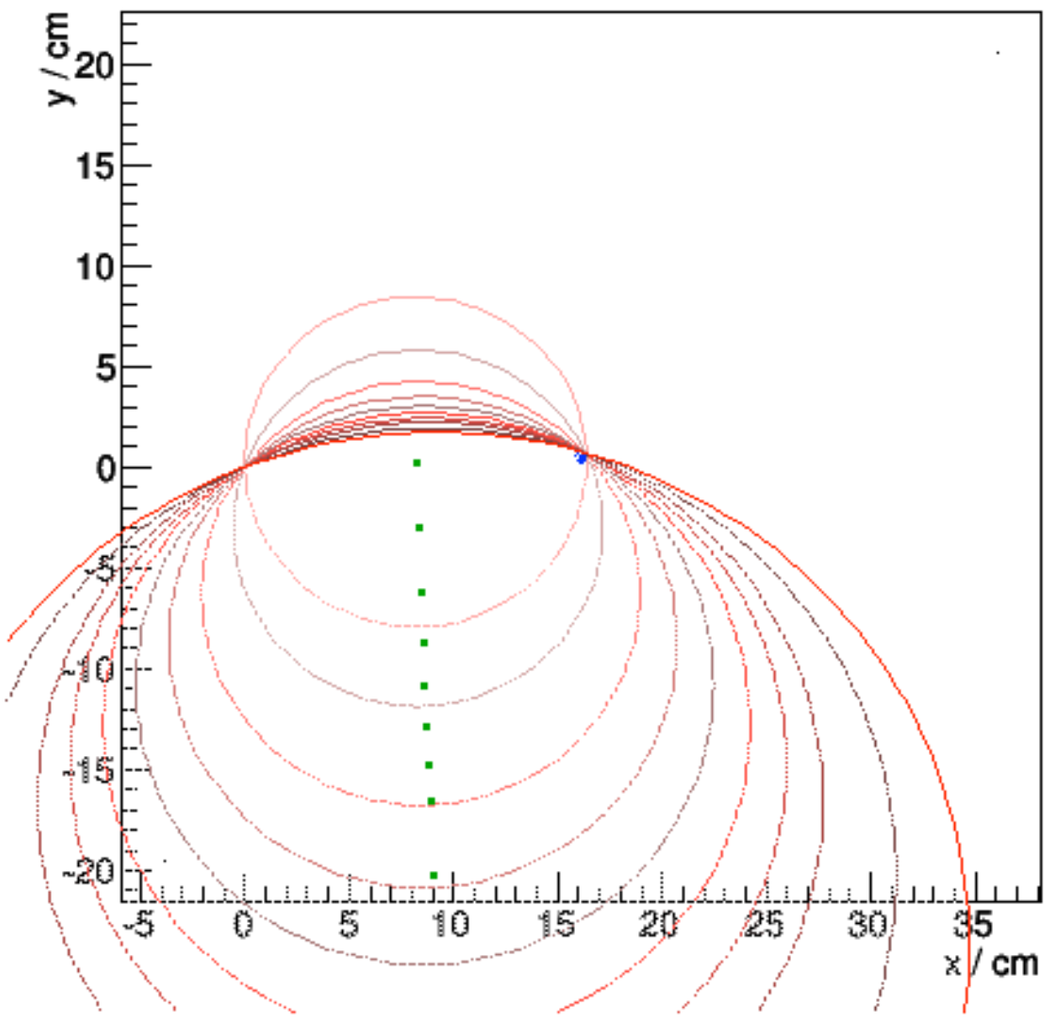
PANDA STT+MVD

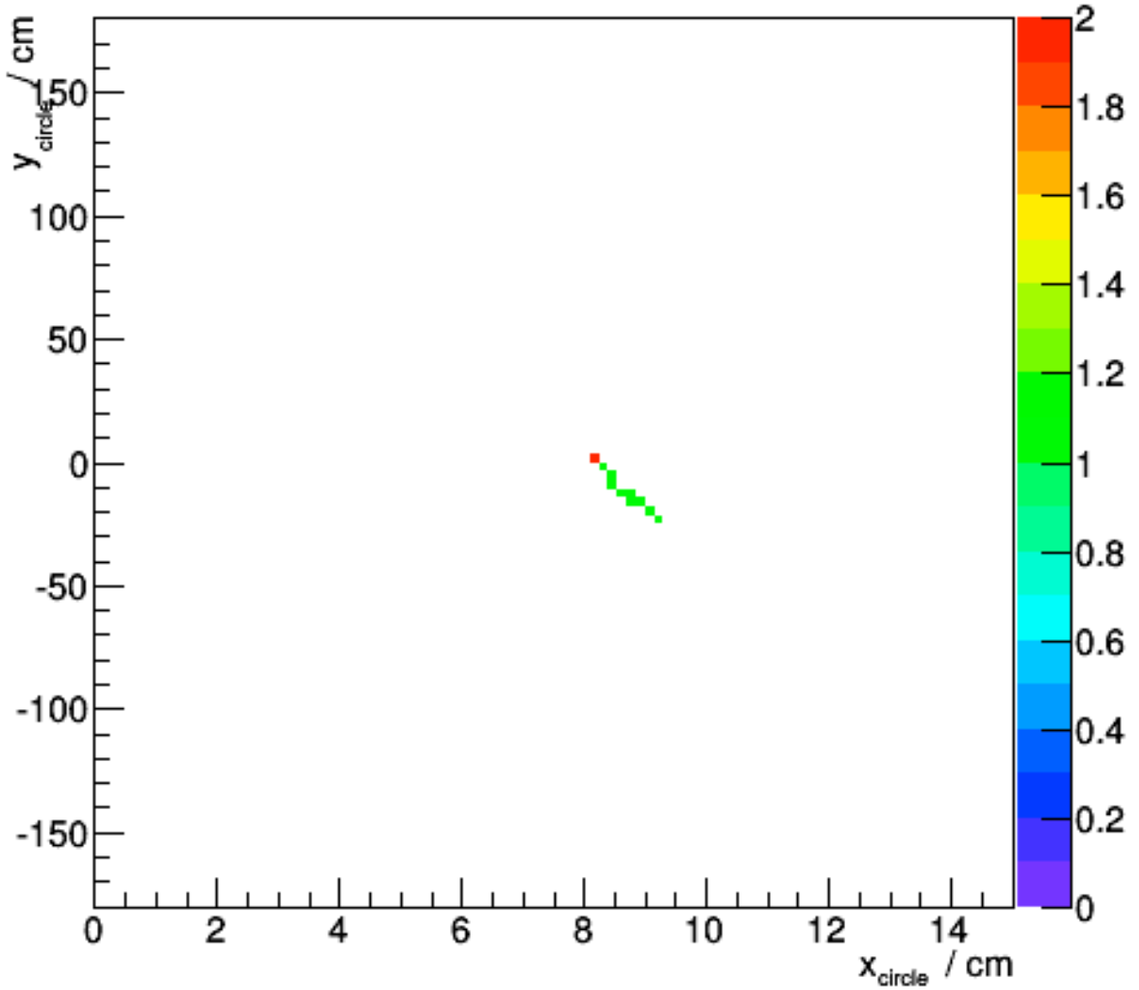
1800 x 1800 Grid

Hough Transform — Variations

- **Circle** Hough with isochrones

static slide





Circle Hough — GPU @ PandaRoot

- Built core functions in CUDA
- Integrated into PandaRoot code by means of interfacer
- Parallelism on a per-Hough calculation basis

  more possible

Circle Hough — GPU @ PandaRoot

PndIsochroneTrackFinder.cxx (CPU)

```
TVector IsoCalc(double x0, double y0, double isoR, double phi) {
     double s = (isoR * isoR - x0 * x0 - y0 * y0) /(2 * (x0 * TMath::Cos(phi) + y0 * TMath::Sin(phi) + isoR));
     TVector2 result(x0 + s * TMath::Cos(phi), y0 + s * TMath::Sin(phi));
     return result;
}
```

cudaHough.cu (GPU)

```
__global__ void IsoCalc_Gpu_Kernel(float *angles, float *houghX, float *houghY, int nAlphas, float x, float y, float r) {
     int idx = blockIdx.x * blockDim.x + threadIdx.x;
     if (idx < nAlphas) {
          float sinVal, cosVal;
          sincosf(angles[idx], &sinVal, &cosVal);

          float s = (r * r - x * x - y * y) / (2 * (x * cosVal + y * sinVal + r));
          houghX[idx] = x + s * cosVal;
          houghY[idx] = y + s * sinVal;
     }
}

extern "C" std::vector<TVector2> IsoCalc_Gpu(float * angles_h, int nAlphas, float x, float y, float r) {
     ...
     IsoCalc_Gpu_Kernel<<<nBlocks, nThreads>>>(angles_d, houghX_d, houghY_d, nAlphas, x, y, r); // invoke actual kernel
     ...
}
```

Circle Hough — GPU @ PandaRoot

PndIsochroneTrackFinder.cxx

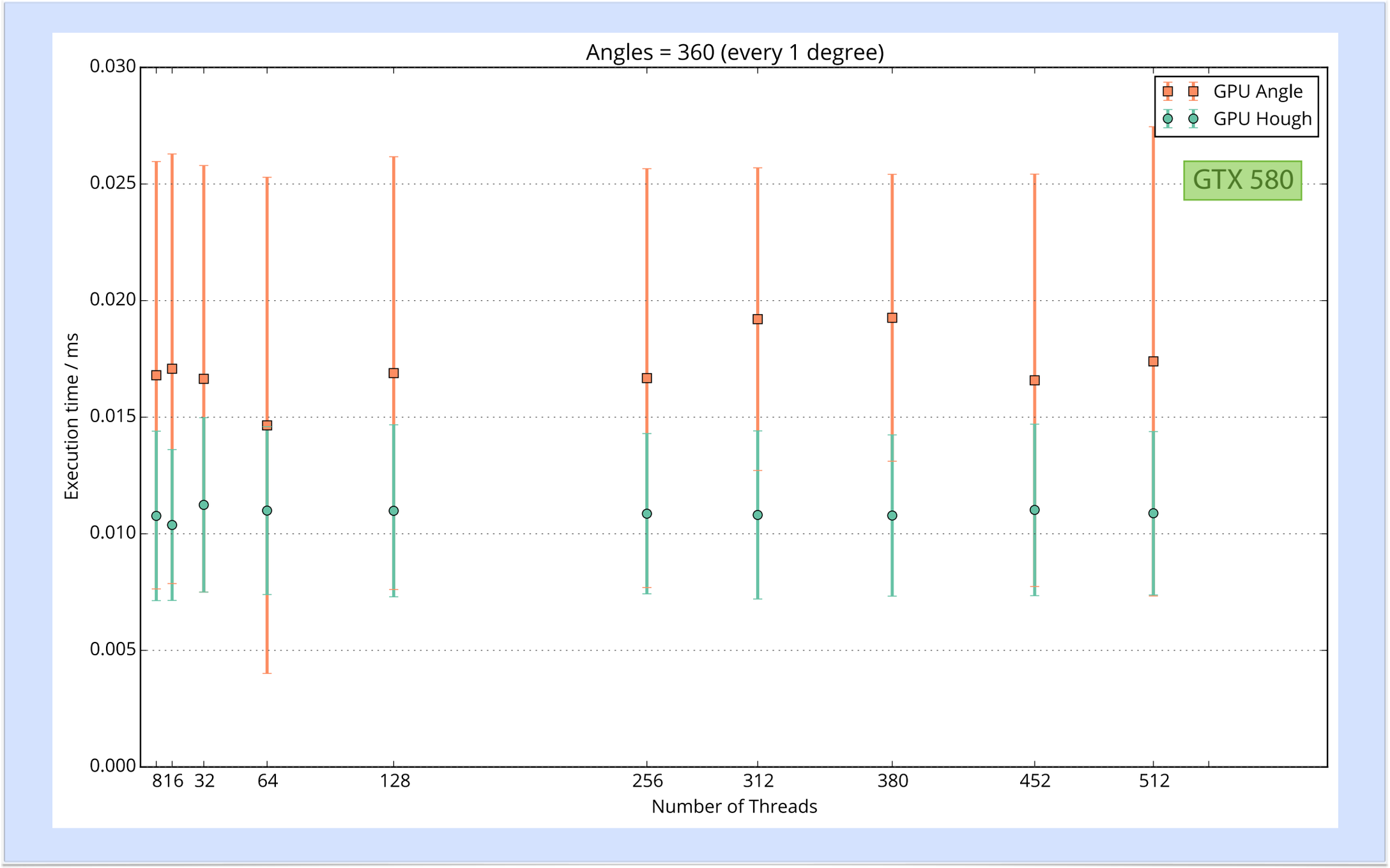
```
PndIsochroneTrackFinderGpuInterfacer * gpuInterfacer;
std::vector<TVector2> circleCenters;
if (fUseGpu) {
     gpuInterfacer = new PndIsochroneTrackFinderGpuInterfacer();
     circleCenters = gpuInterfacer->IsoCalc(xVal, yVal, isoChrone);
} else {
     circleCenters = GenerateHoughValues(angles, xVal, yVal, isoChrone);
}
```

macro.C

```
PndIsochroneTrackFinderTask* isochroneTracker = new PndIsochroneTrackFinderTask();
isochroneTracker->AddHitBranch("MVDHitsStrip");
isochroneTracker->AddHitBranch("STTHit");
isochroneTracker->SetUseGpu(true);
```

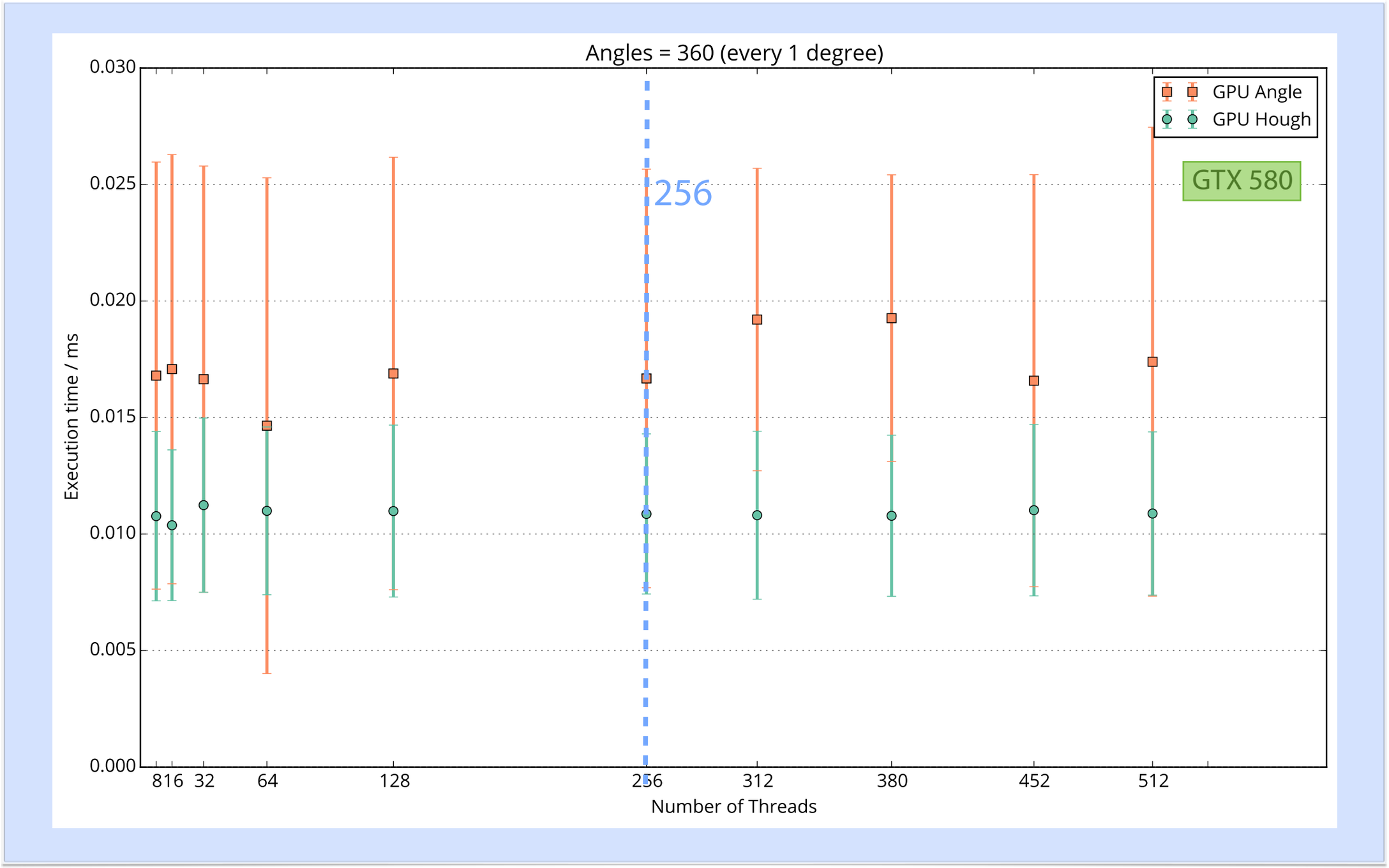
Circle Hough — Performances

Variation of number of threads (and blocks): GPU values



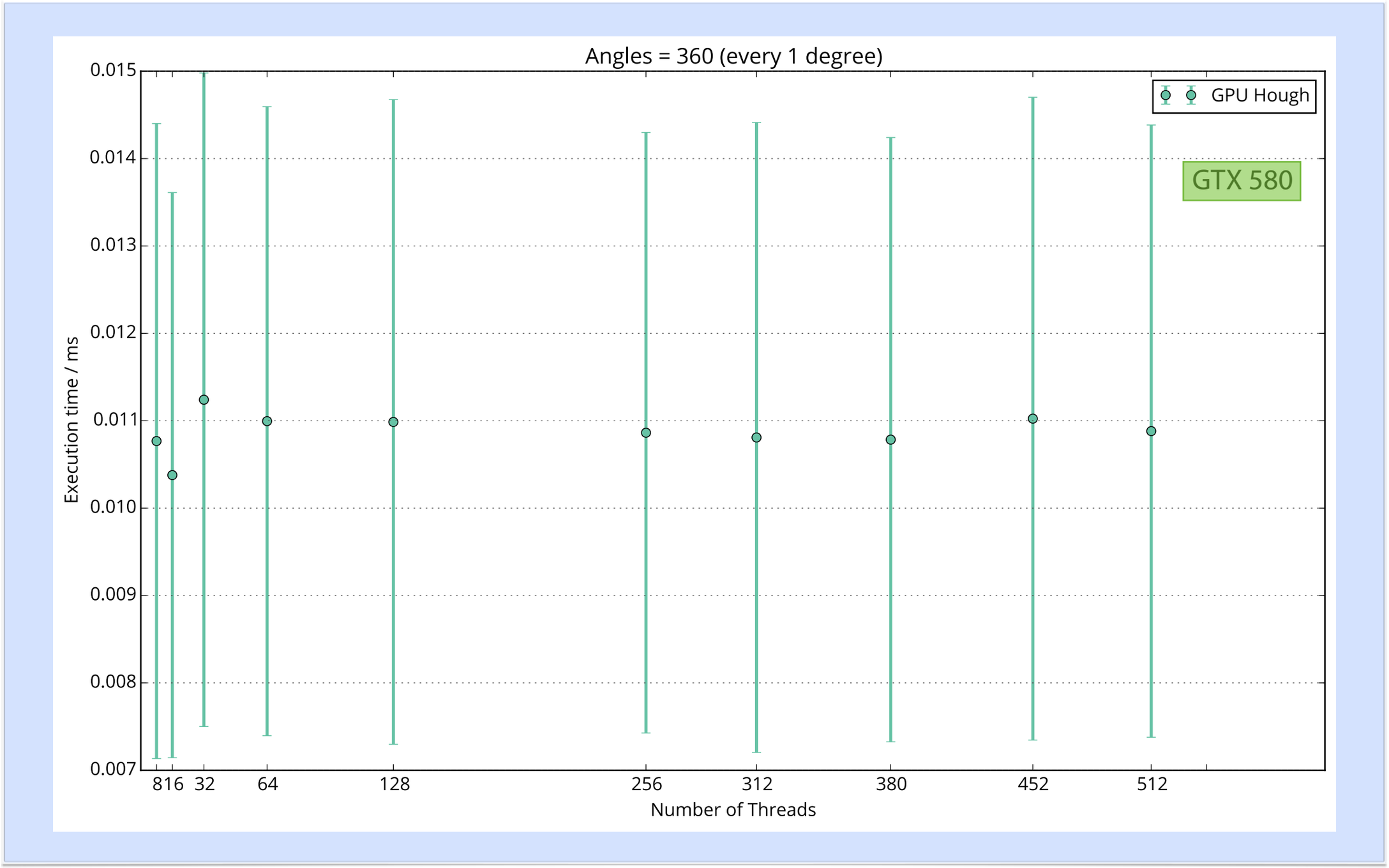
Circle Hough — Performances

Variation of number of threads (and blocks): GPU values



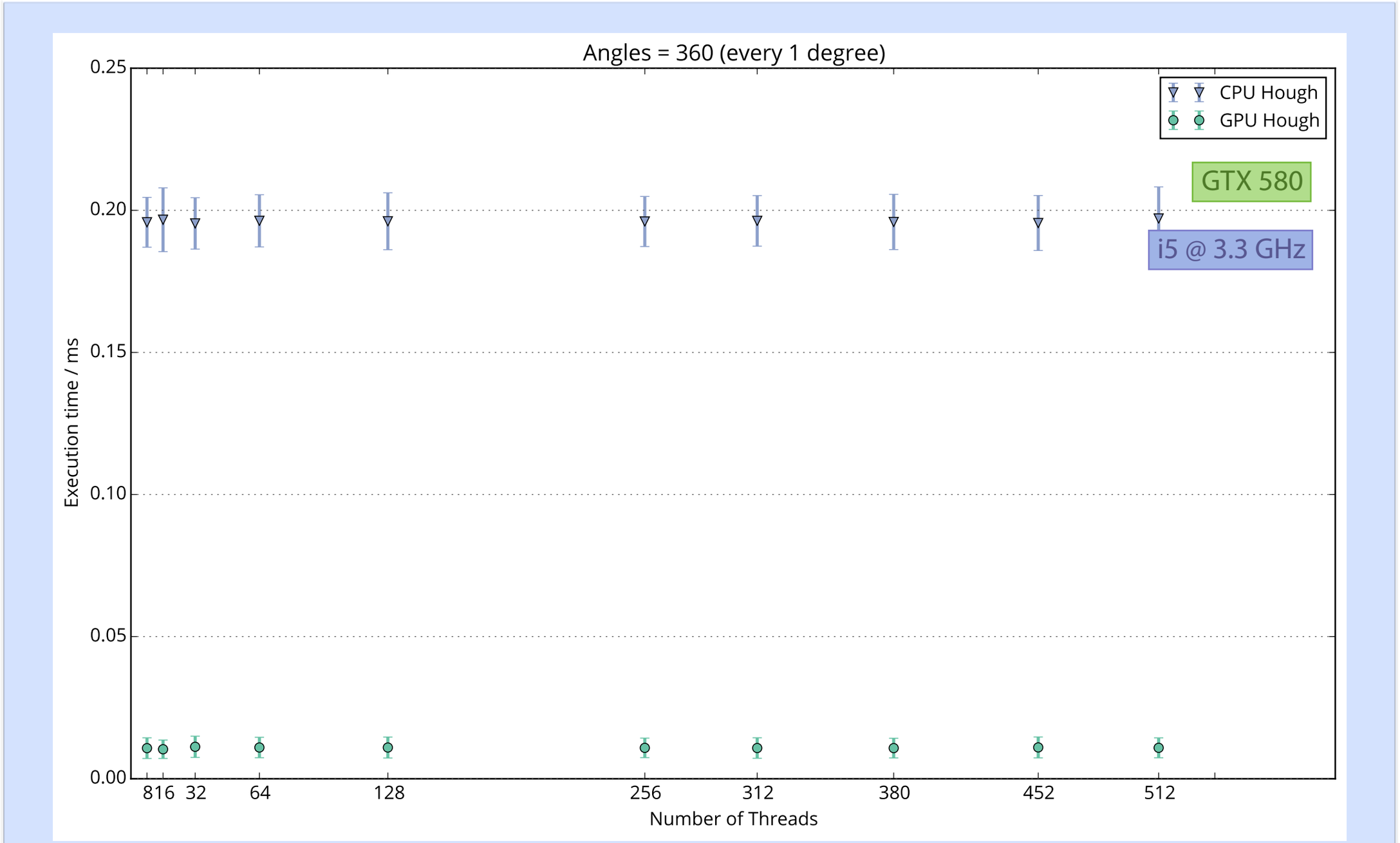
Circle Hough — Performances

Variation of number of threads (and blocks): GPU vs. CPU



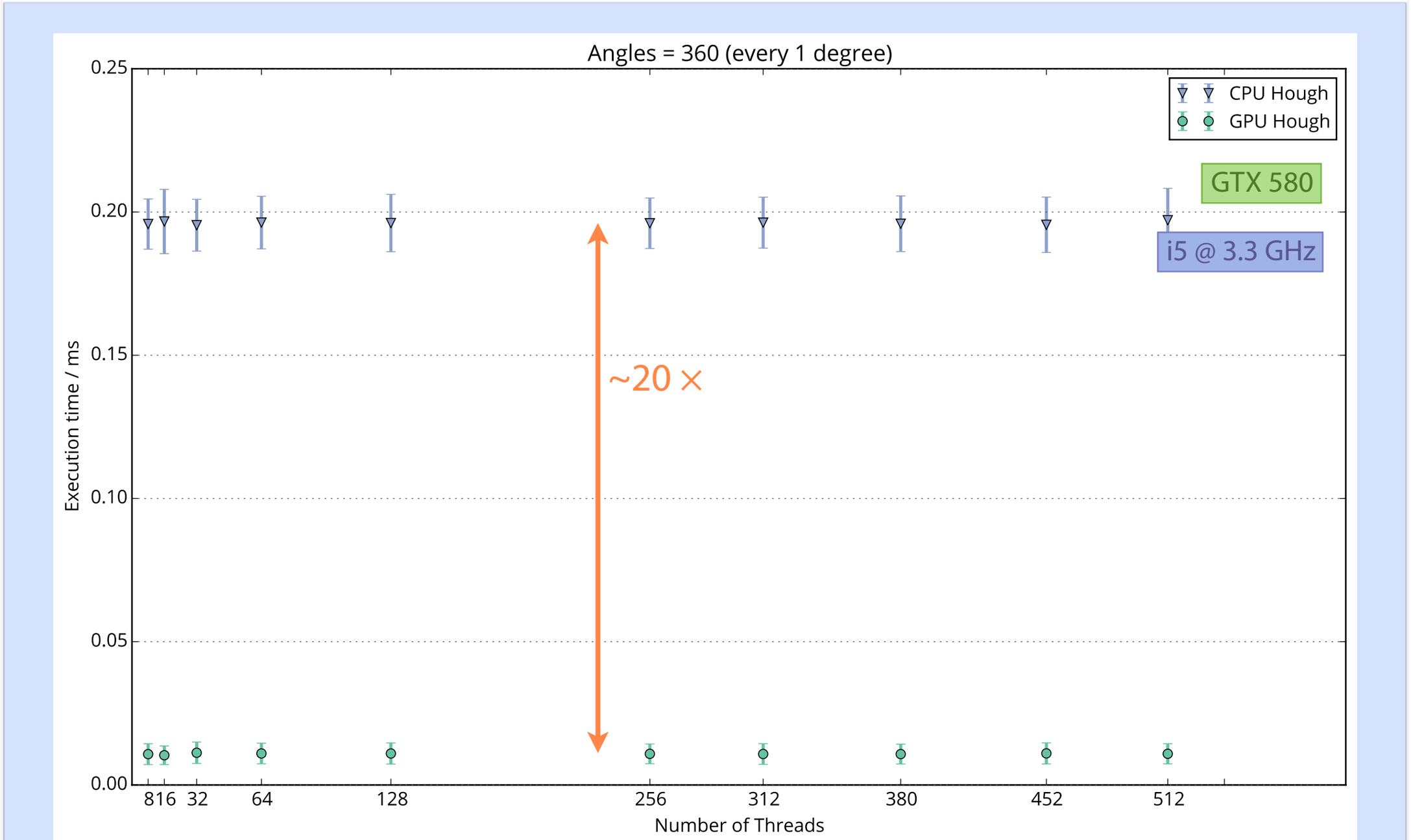
Circle Hough — Performances

Variation of number of threads (and blocks): GPU vs. CPU



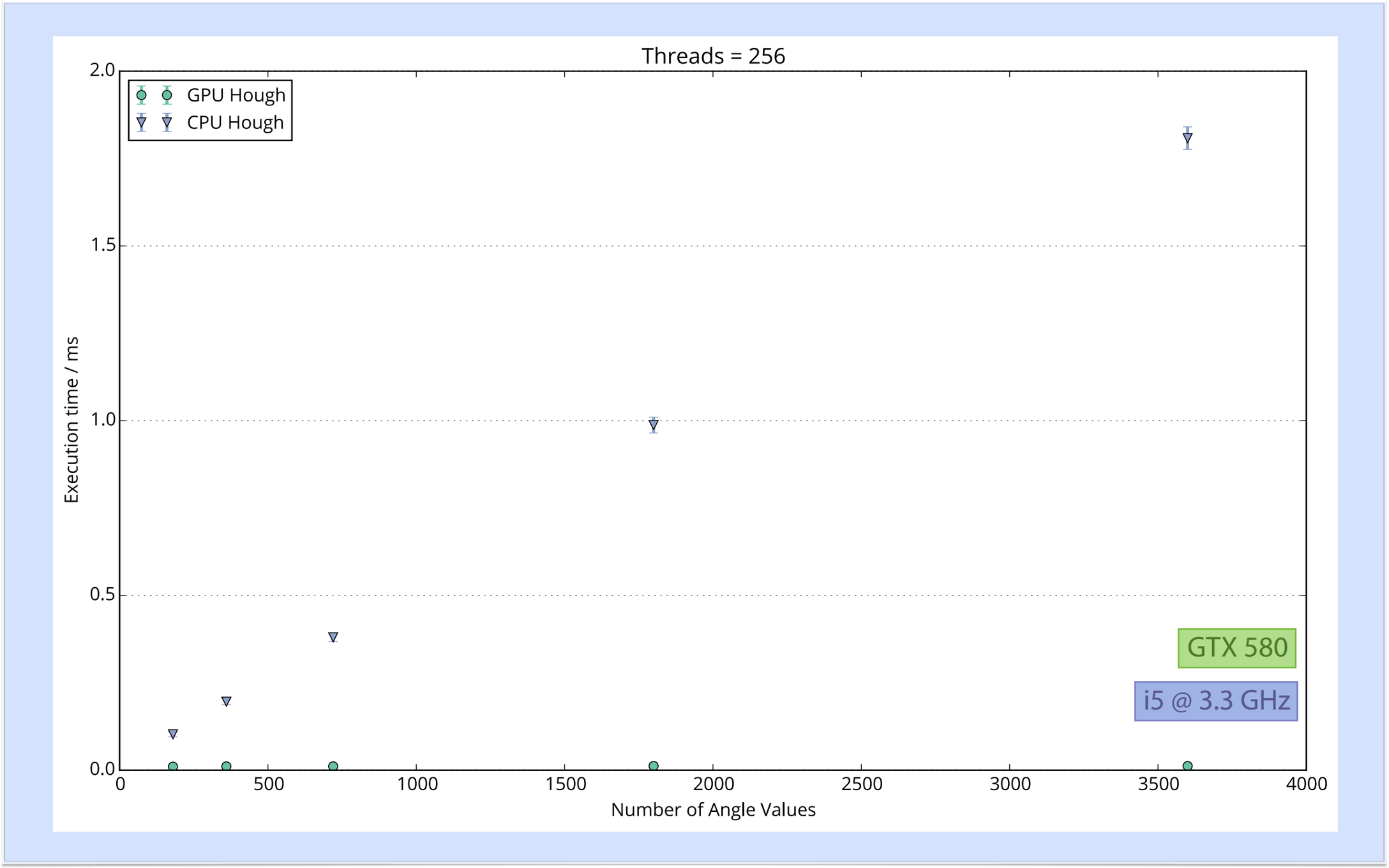
Circle Hough — Performances

Variation of number of threads (and blocks): GPU vs. CPU



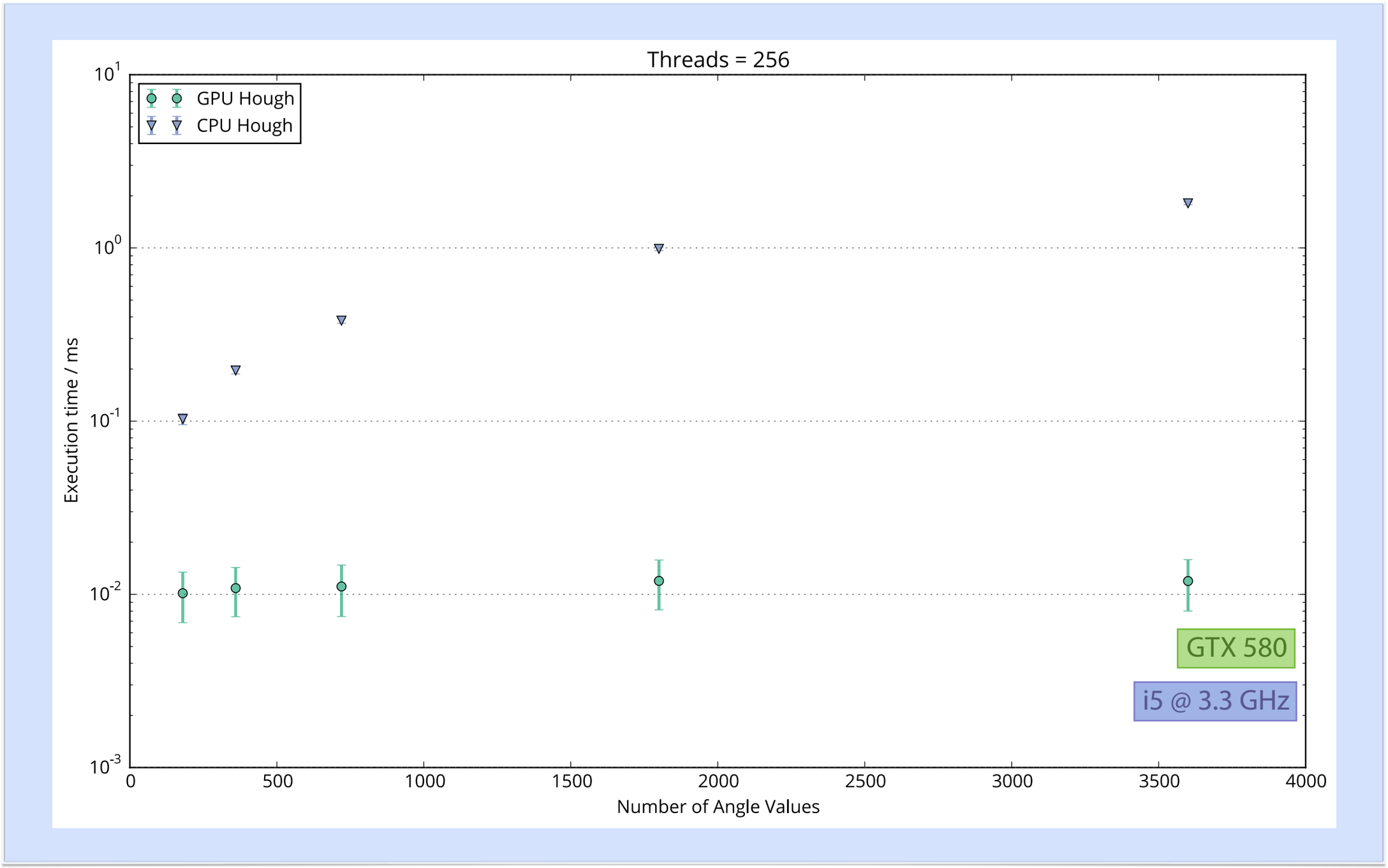
Circle Hough — Performances

Variation of number of angles: GPU vs. CPU



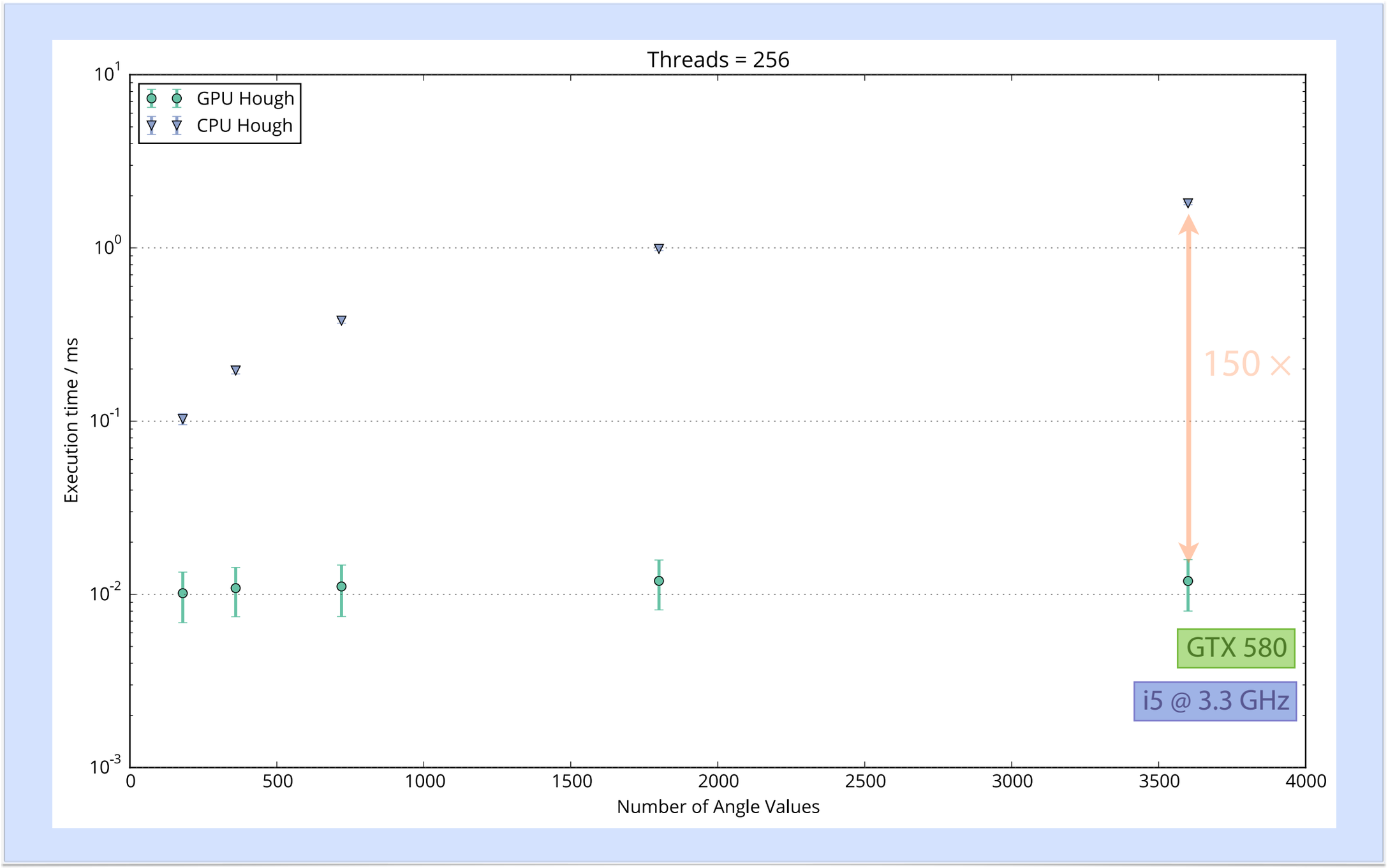
Circle Hough — Performances

Variation of number of angles: GPU vs. CPU



Circle Hough — Performances

Variation of number of angles: GPU vs. CPU



JÜLICH
FORSCHUNGSZENTRUM

panda

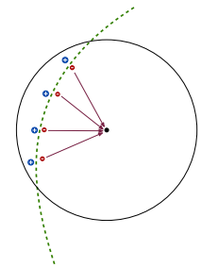
ALGORITHMS #2

Hough Transform

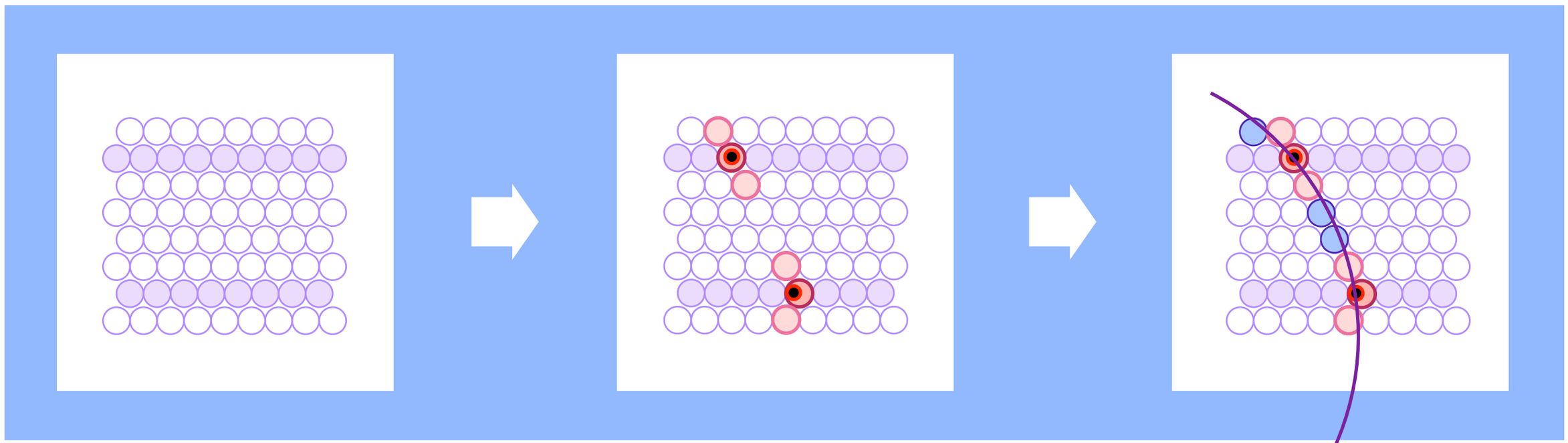
Triplet Finder

Triplet Finder

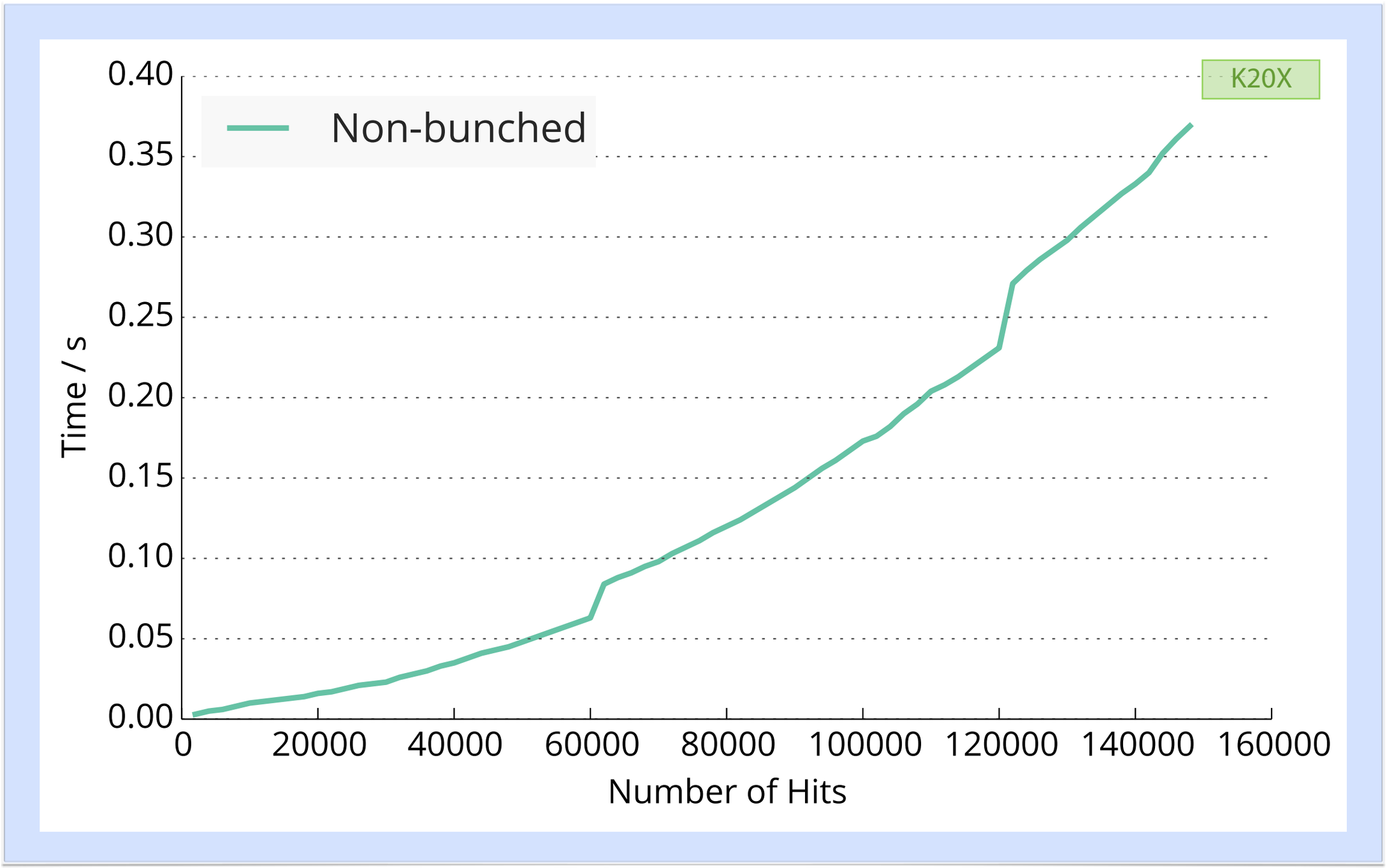
- ***Idea:*** Use only subset of detector as seed
  - Don‘t use STT isochrones *(drift times)*



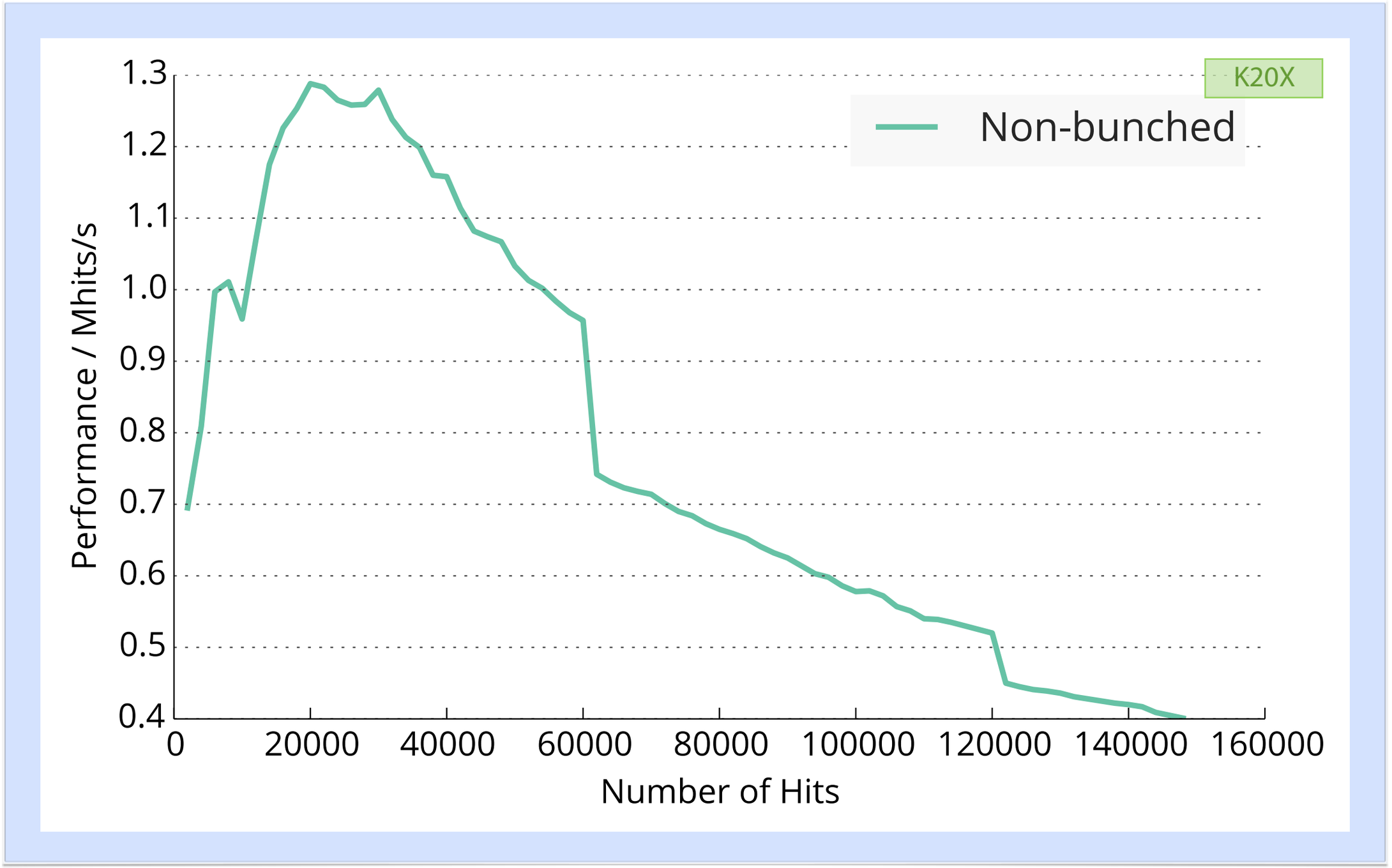
- Features
  - Fast & robust algorithm, no event time needed
  - Many tuning possibilities
- Optimizations by NVIDIA Application Lab (Andrew)



Triplet Finder — Times



Triplet Finder — Times



Triplet Finder — Optimizations

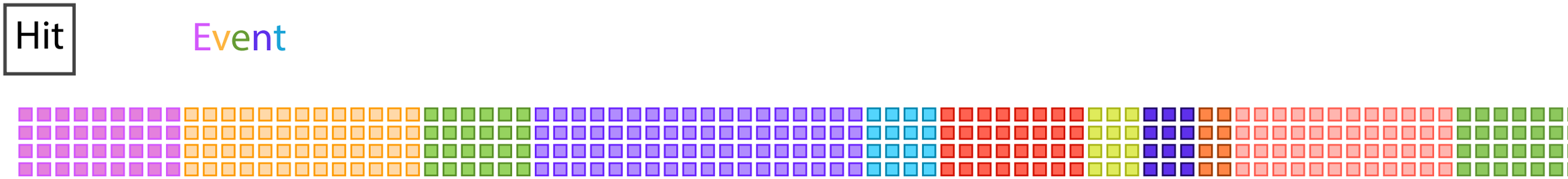
- Bunching Wrapper
  - Hits from one event have similar timestamps
  - Combine hits to sets (bunches) which occupy GPU best

Triplet Finder — Optimizations

- Bunching Wrapper
  - Hits from one event have similar timestamps
  - Combine hits to sets (bunches) which occupy GPU best

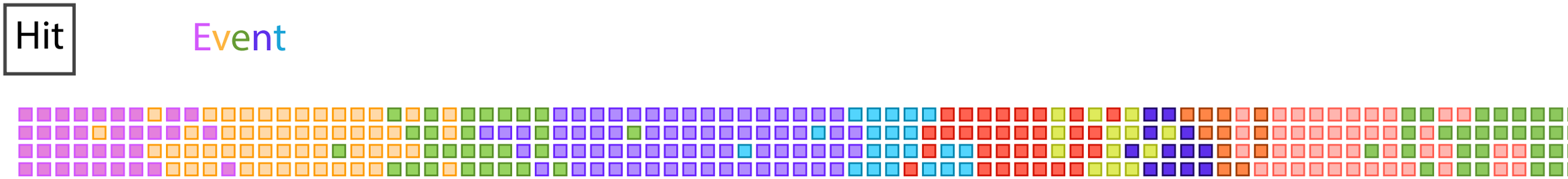
Triplet Finder — Optimizations

- Bunching Wrapper
  - Hits from one event have similar timestamps
  - Combine hits to sets (bunches) which occupy GPU best



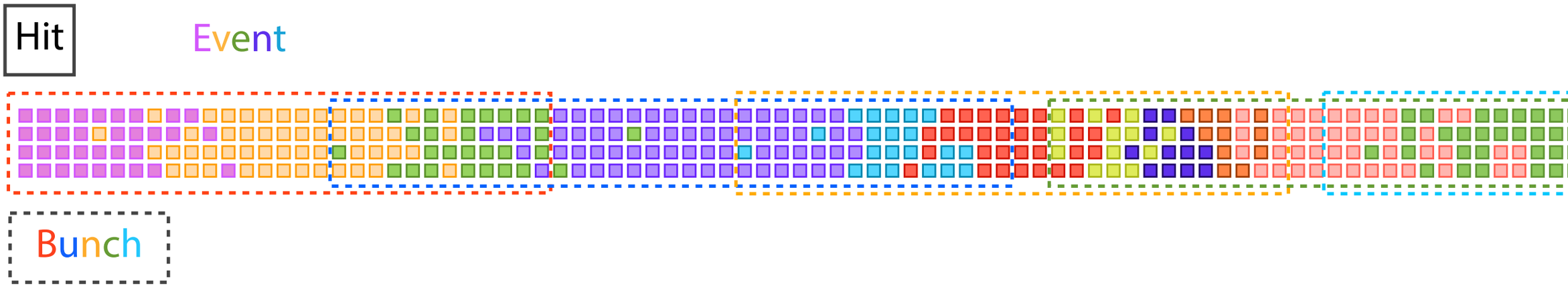
Triplet Finder — Optimizations

- Bunching Wrapper
  - Hits from one event have similar timestamps
  - Combine hits to sets (bunches) which occupy GPU best



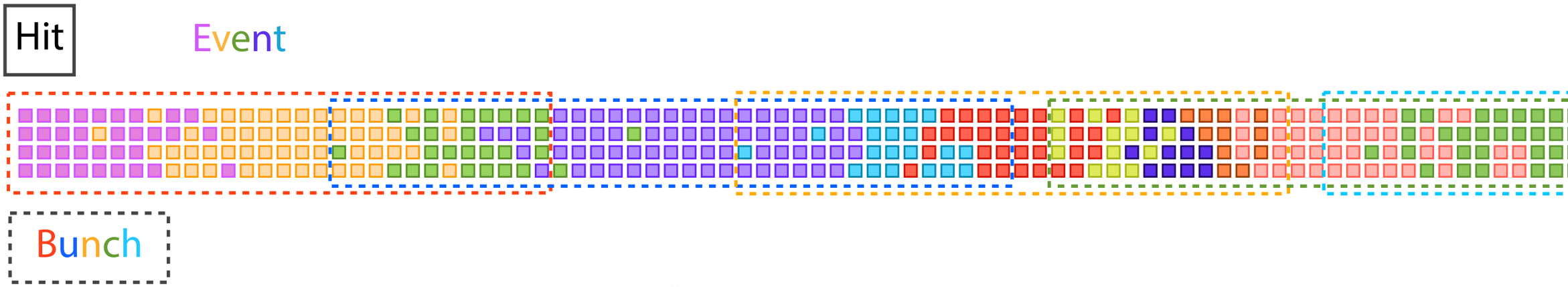
Triplet Finder — Optimizations

- Bunching Wrapper
  - Hits from one event have similar timestamps
  - Combine hits to sets (bunches) which occupy GPU best

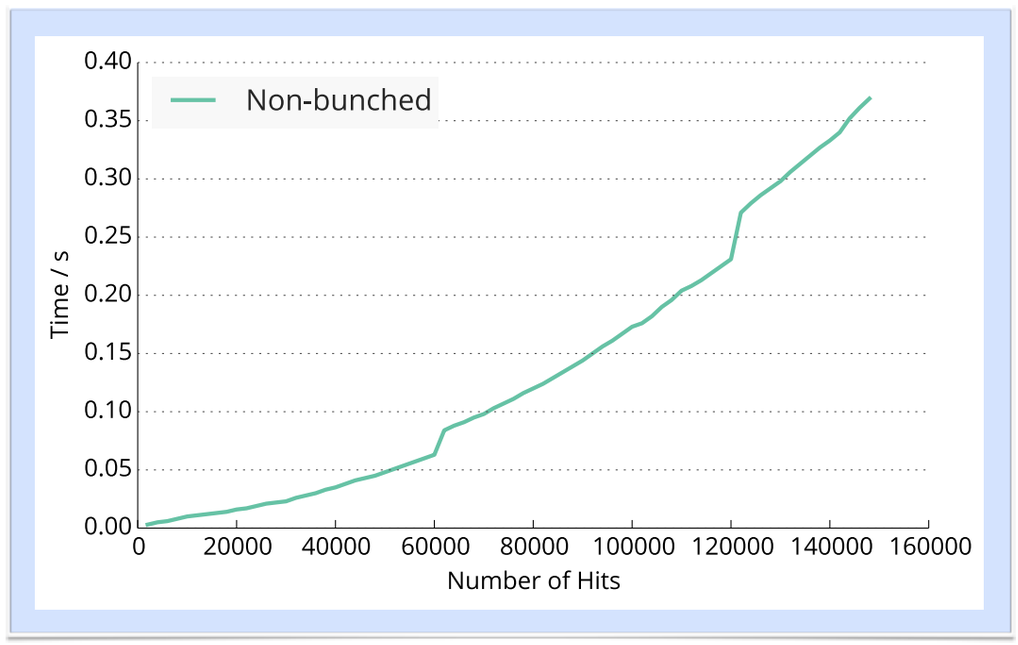


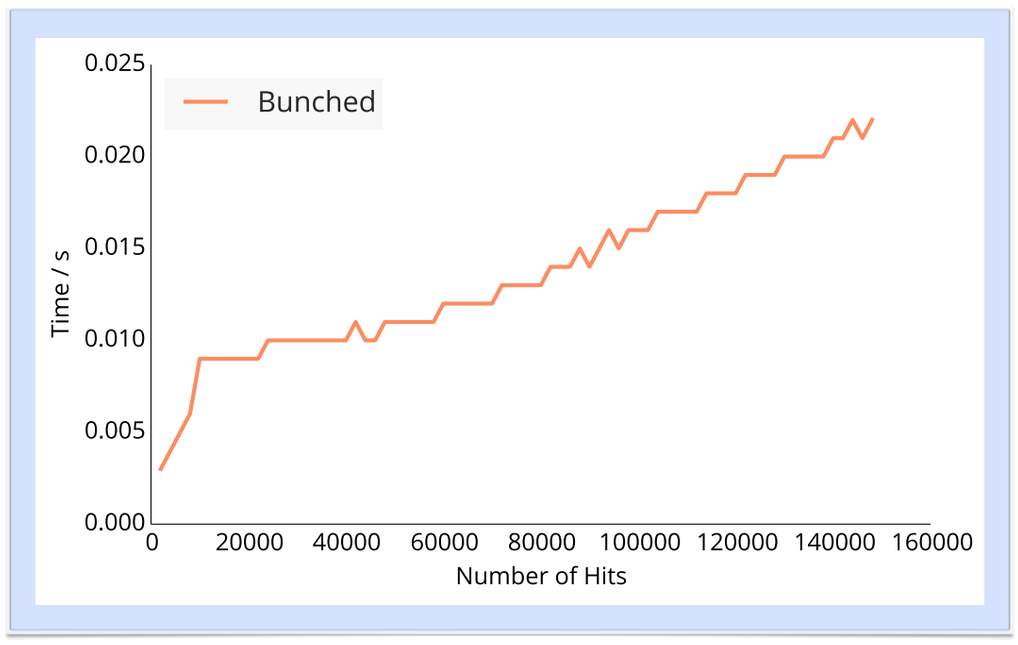
Triplet Finder — Optimizations

- Bunching Wrapper
  - Hits from one event have similar timestamps
  - Combine hits to sets (bunches) which occupy GPU best



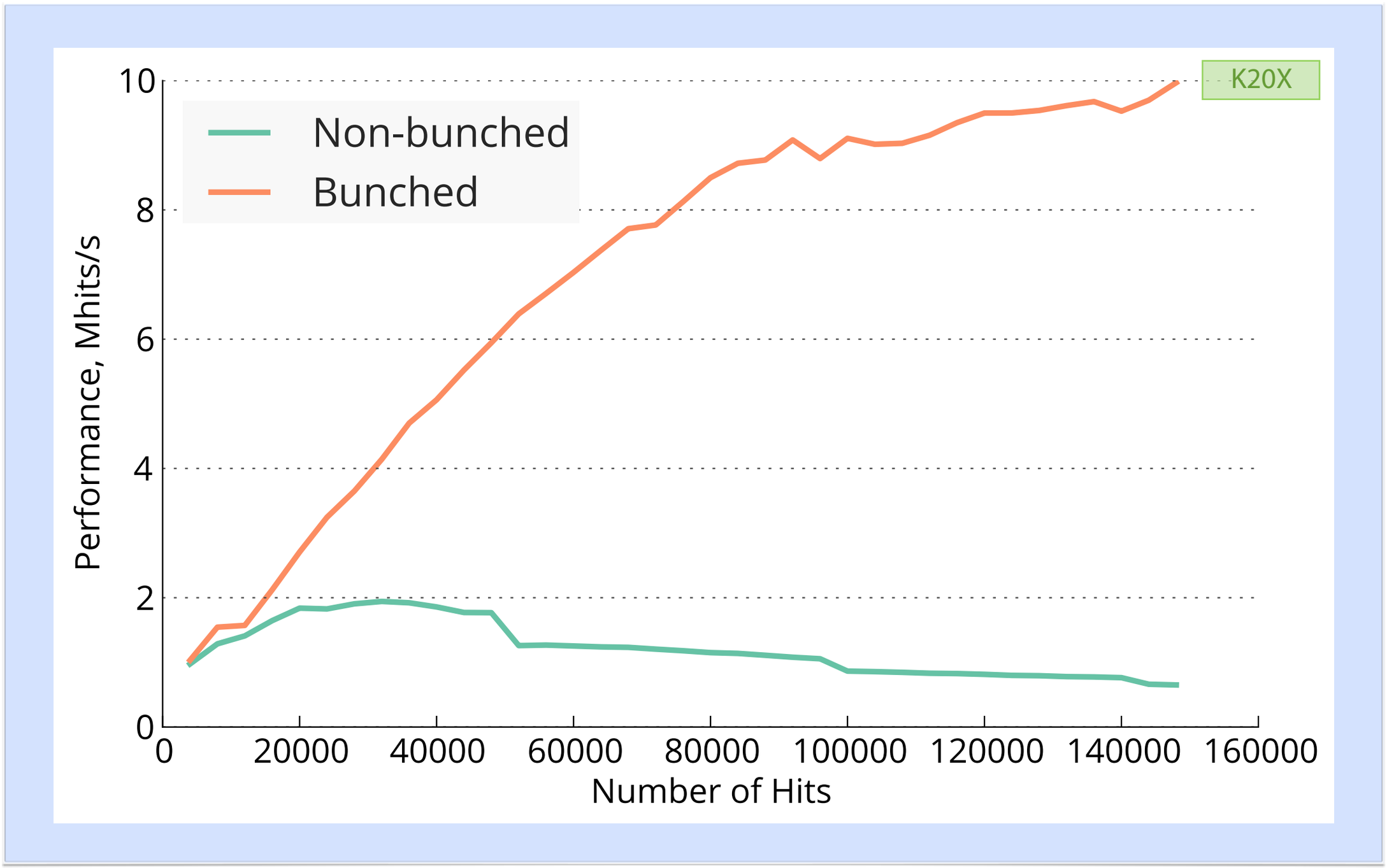
$$\mathcal{O}(N^2) \rightarrow \mathcal{O}(N)$$





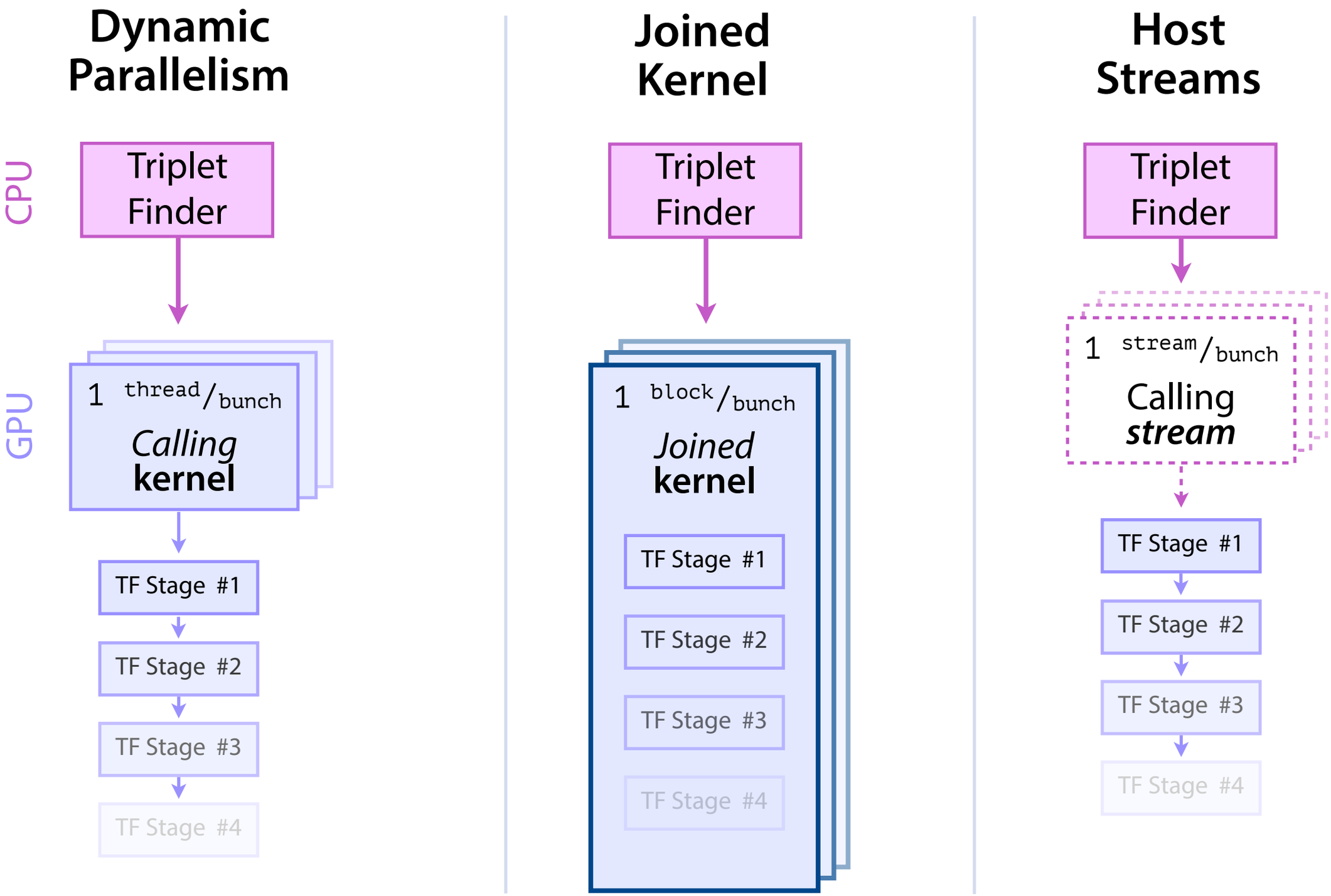
Triplet Finder — Bunching

Performance



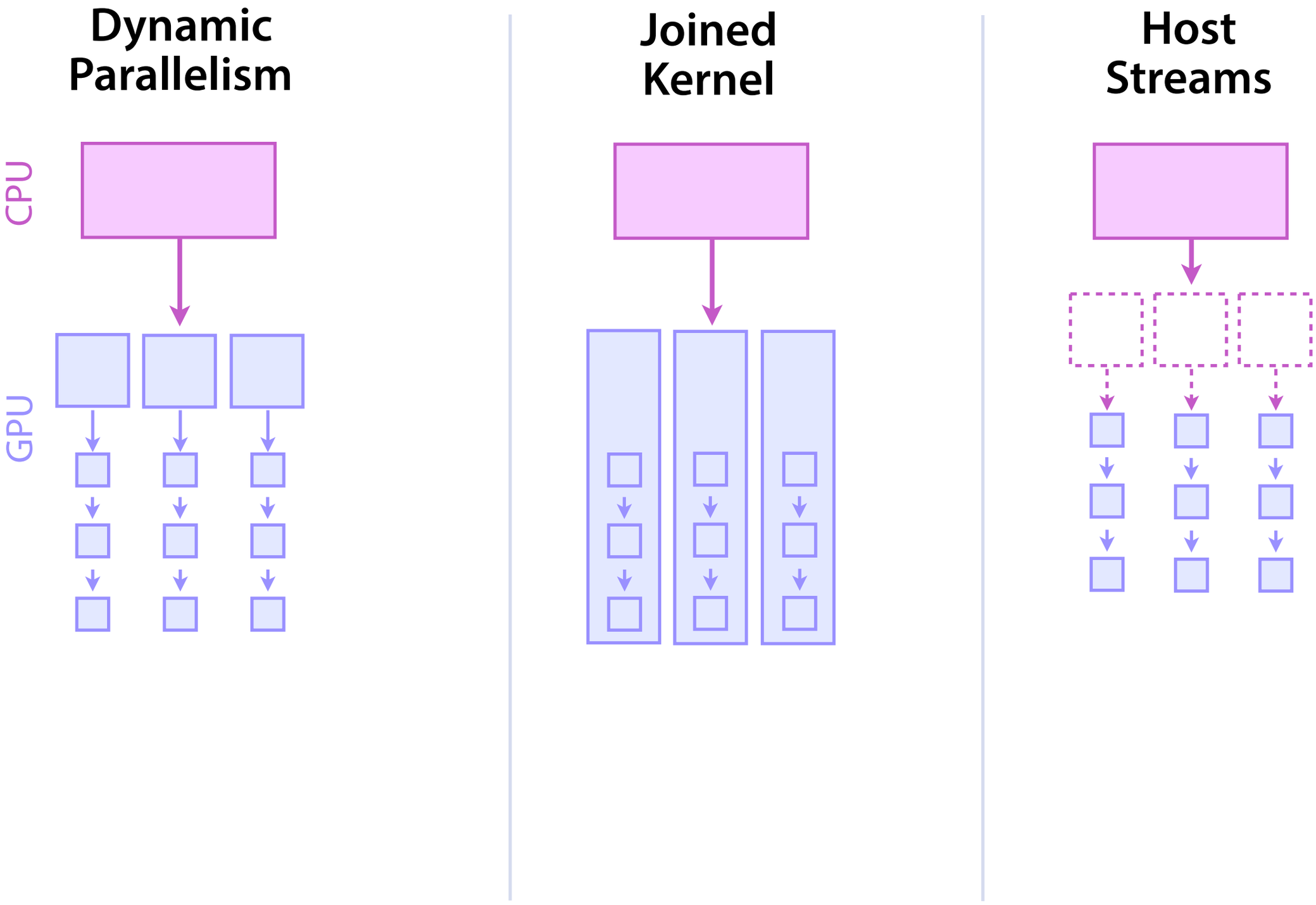
Triplet Finder — Optimizations

- Compare data processing strategies

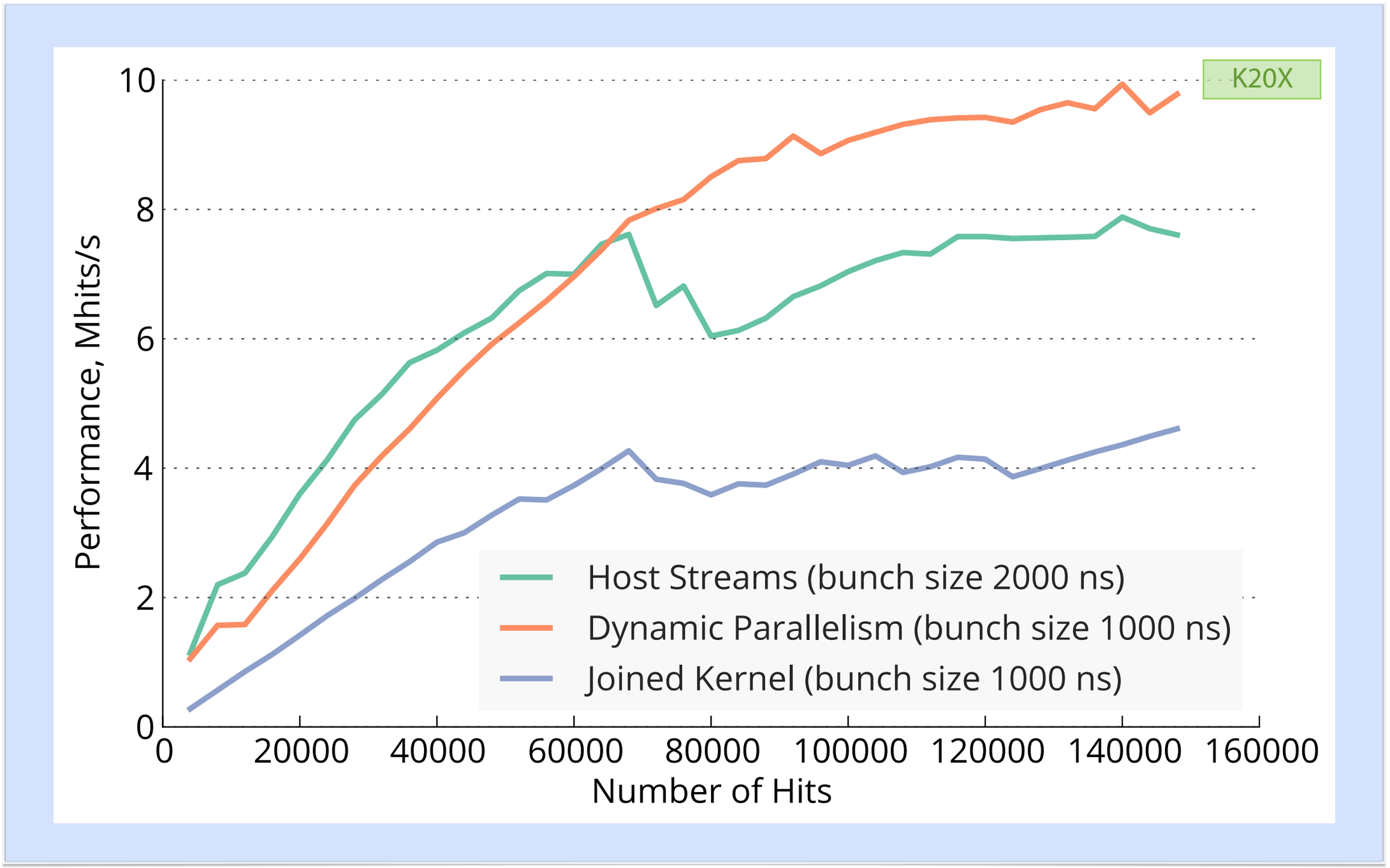


Triplet Finder — Optimizations

- Compare data processing strategies

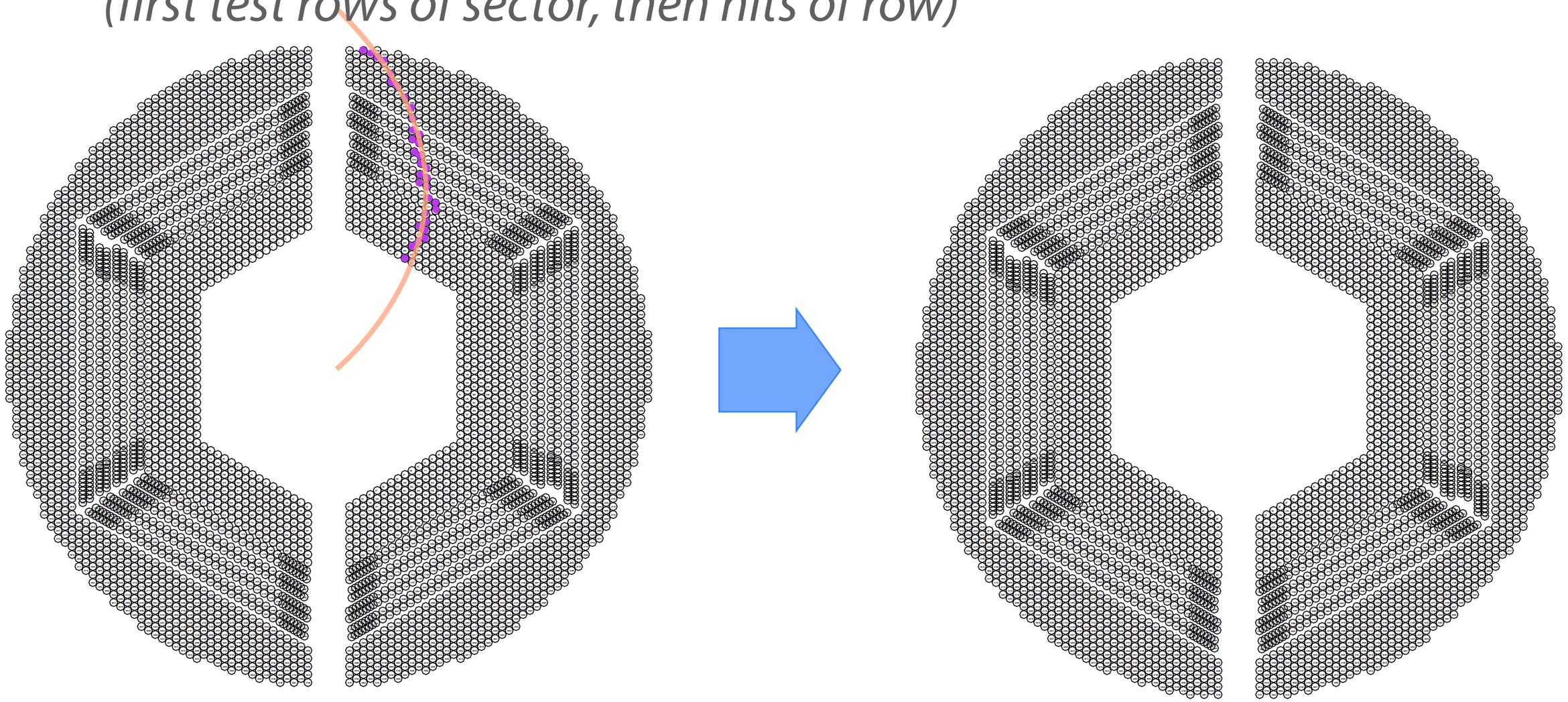


Triplet Finder — Data Processing



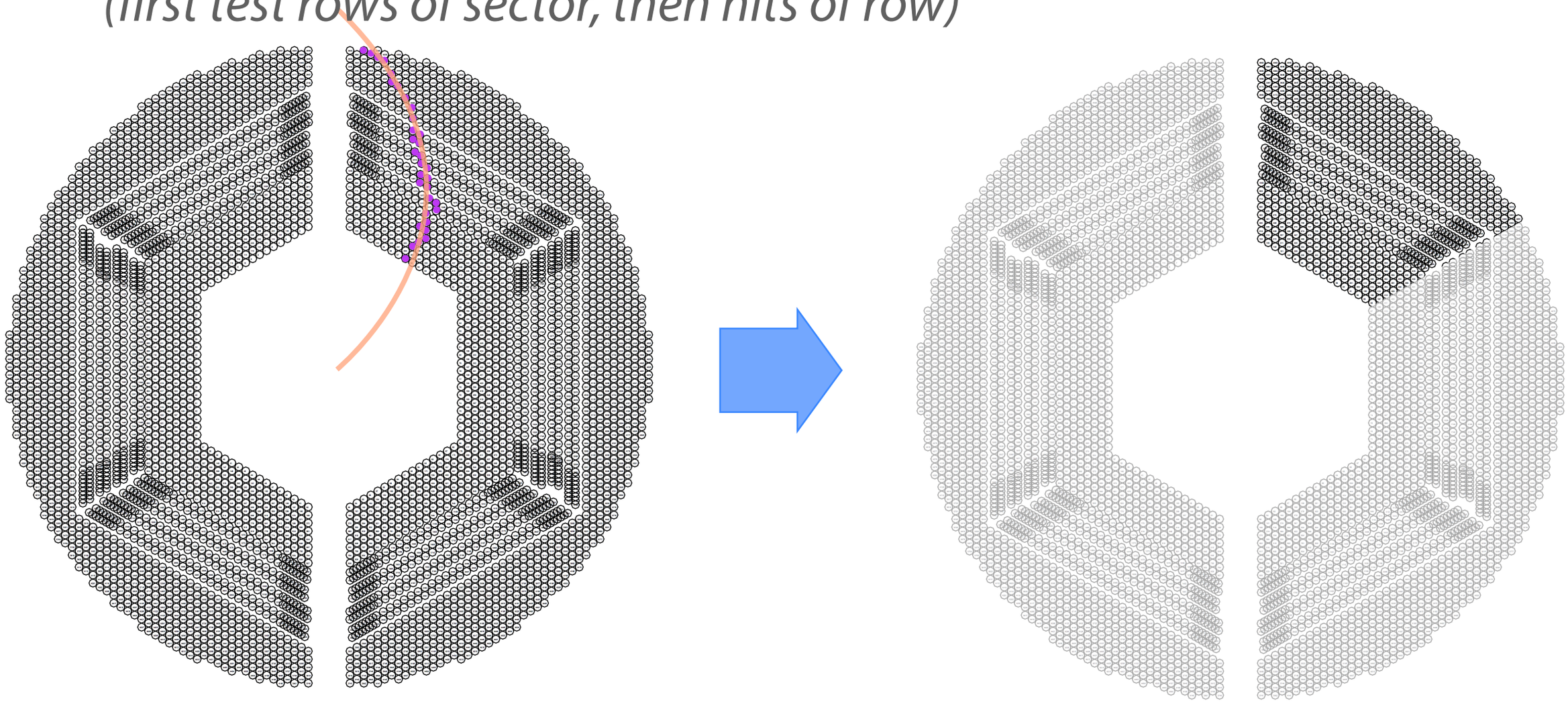
Triplet Finder — Binning: Sector Rows

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



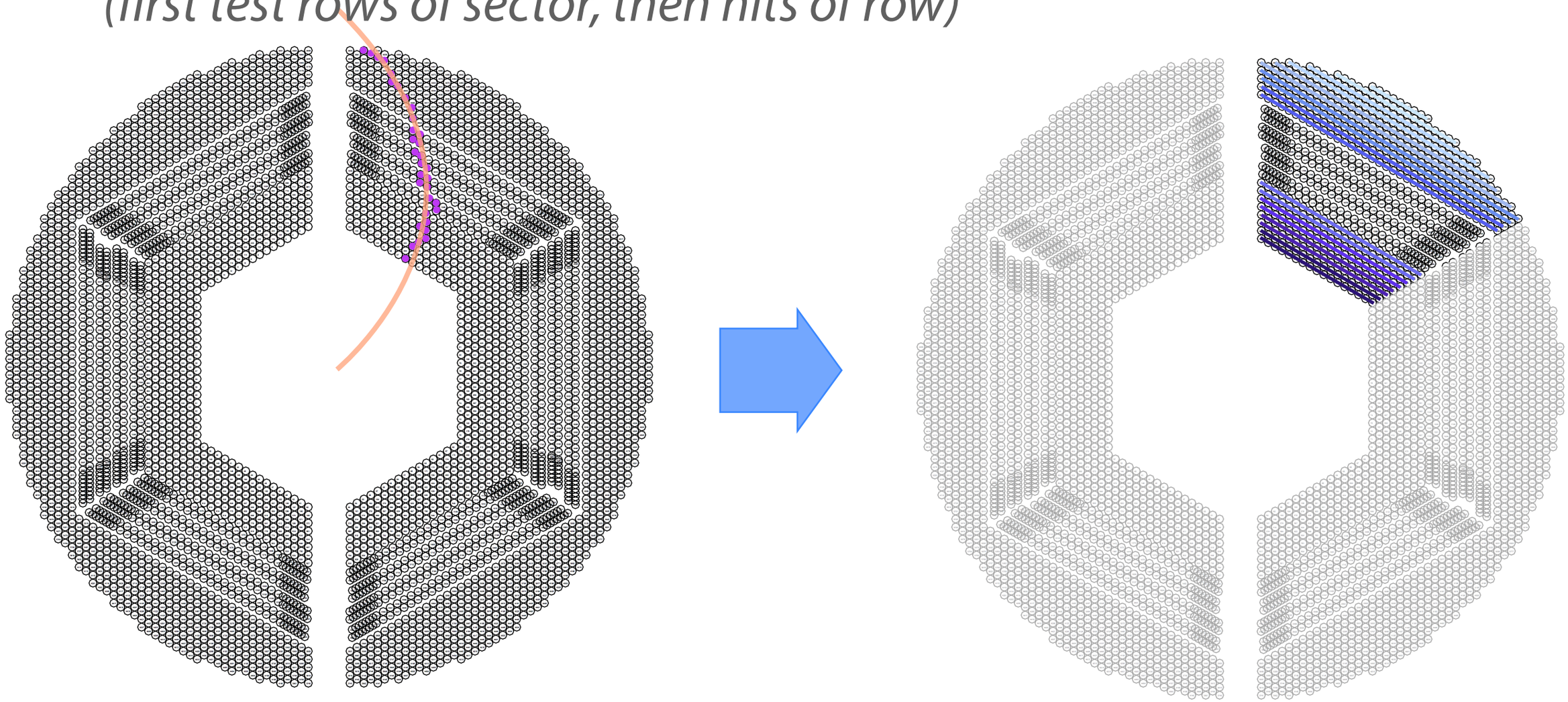
Triplet Finder — Binning: Sector Rows

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



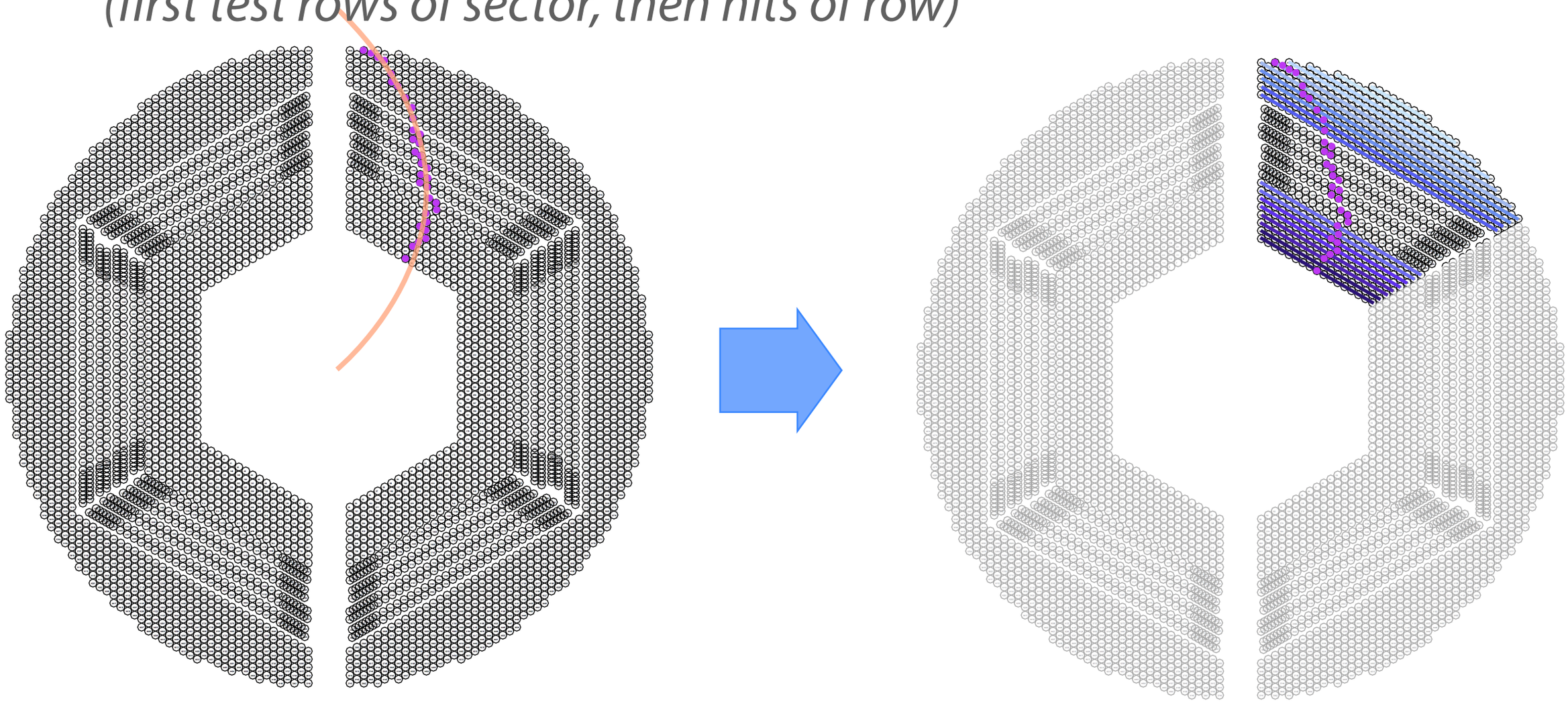
Triplet Finder — Binning: Sector Rows

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



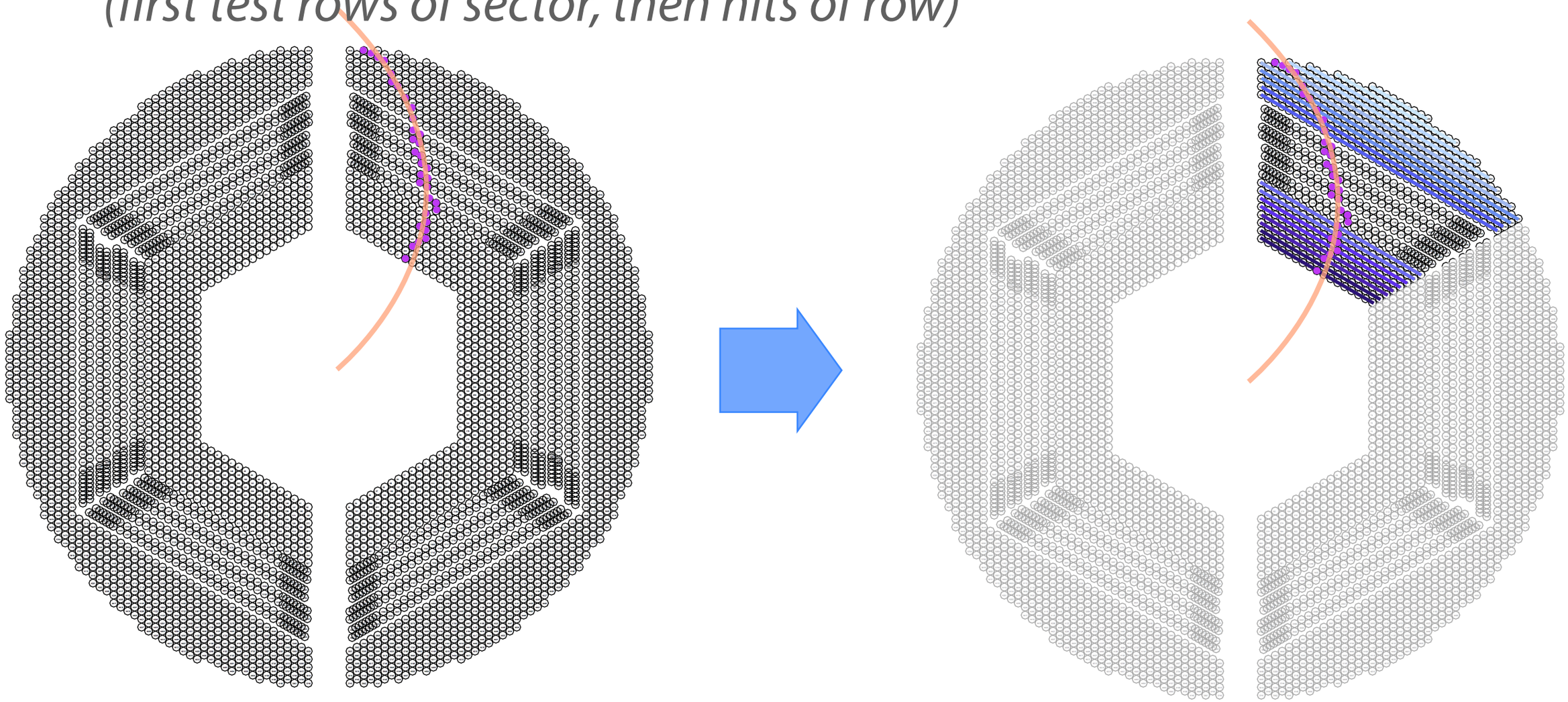
Triplet Finder — Binning: Sector Rows

- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)

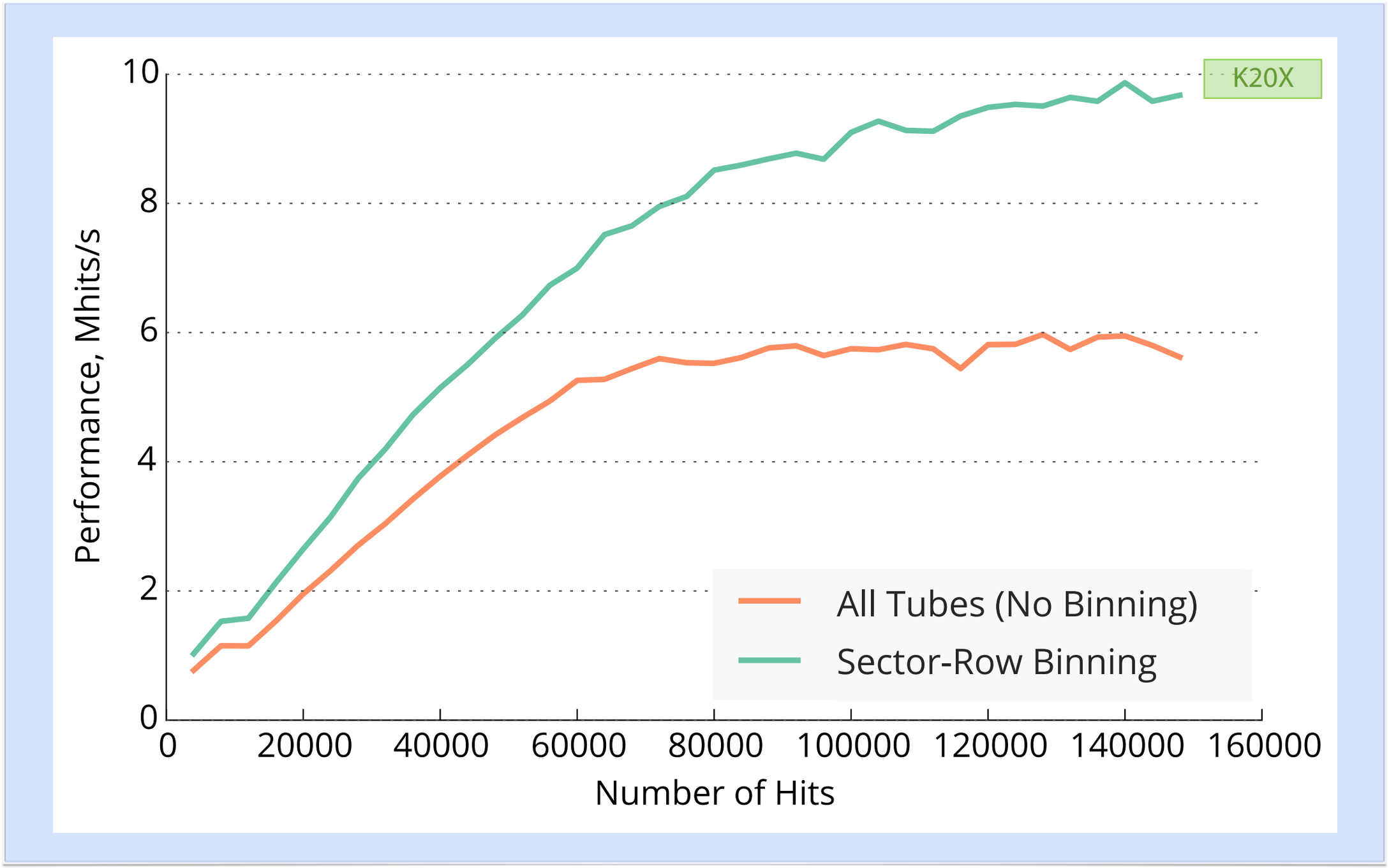


Triplet Finder — Binning: Sector Rows

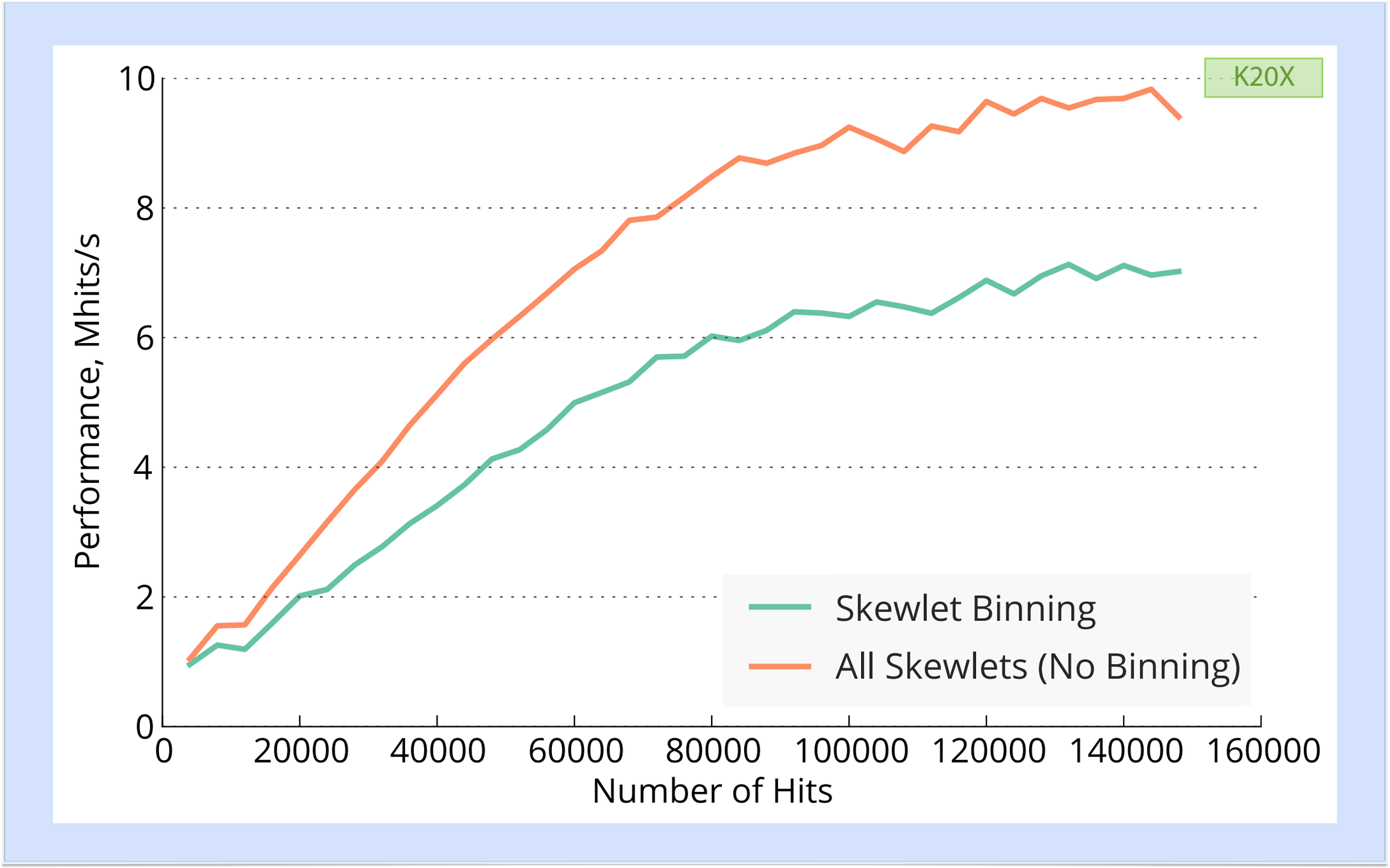
- Sector Row testing
  - After found track:
    Hit association not with *all* hits of current window,
    but only with subset
    (first test rows of sector, then hits of row)



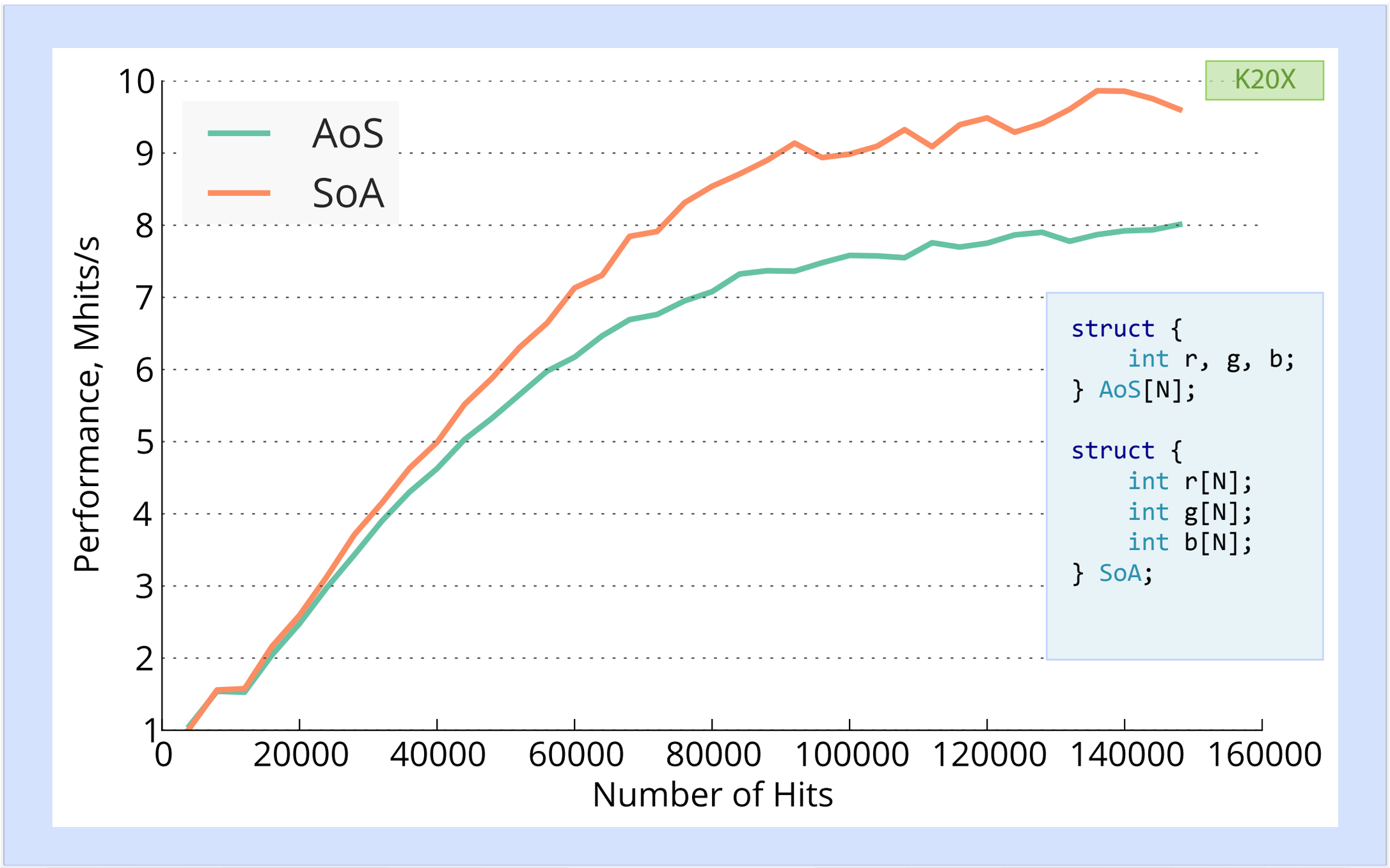
Triplet Finder — Binning: Sector Rows



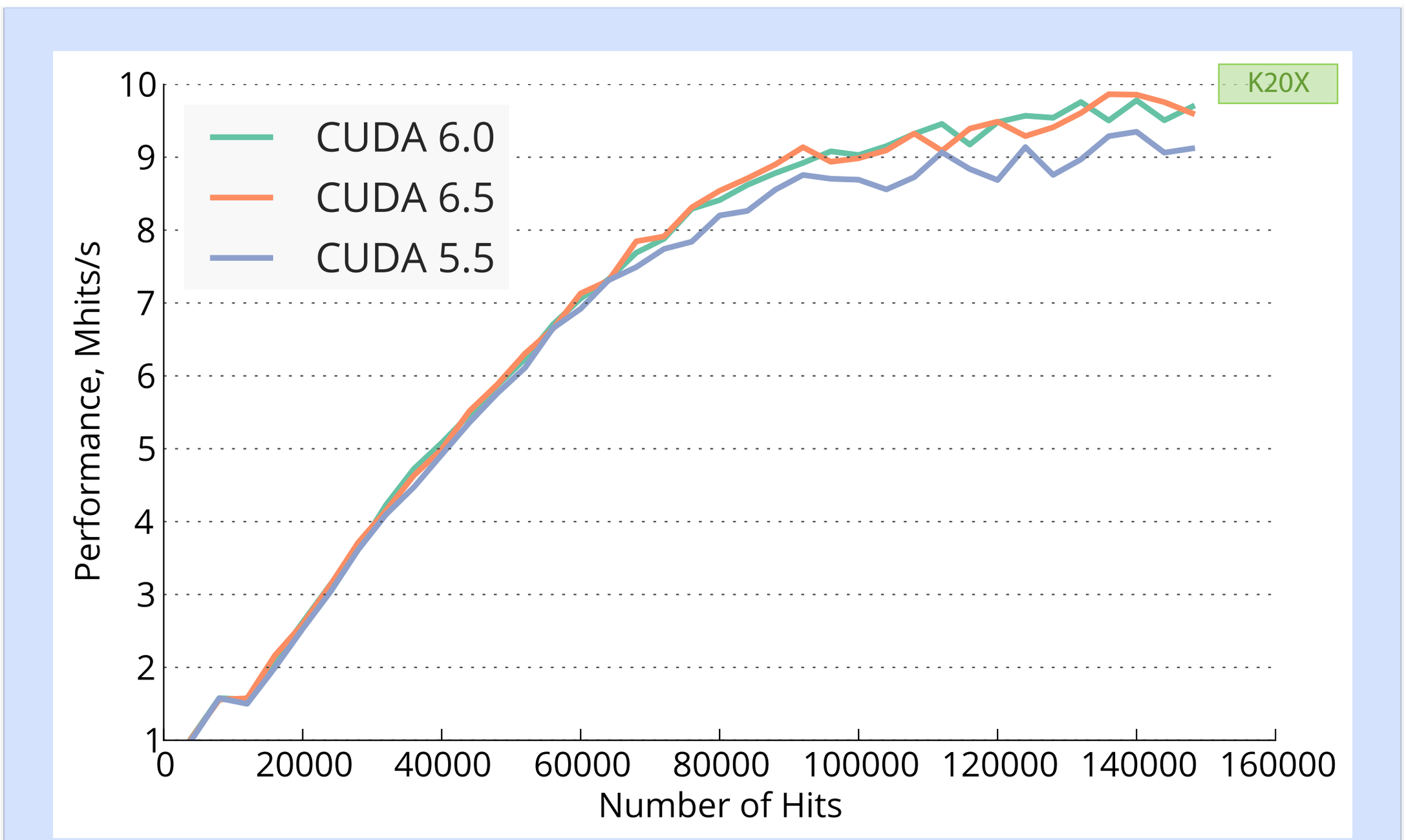
Triplet Finder — Binning: Skewlets



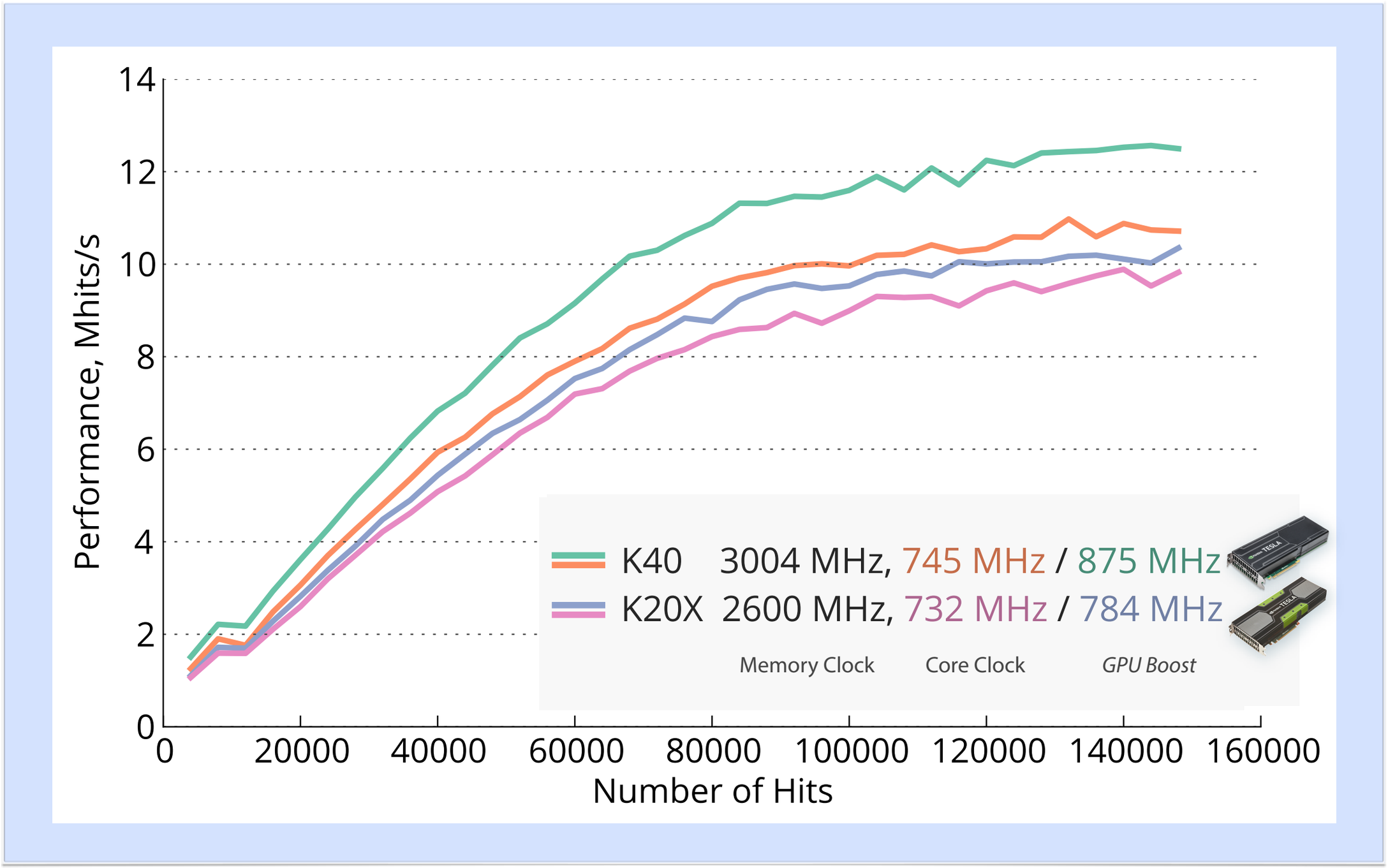
Triplet Finder — AoS vs. SoA



Triplet Finder — CUDA Versions



Triplet Finder — Clock Speed / GPU



Triplet Finder — Comparison to Kepler

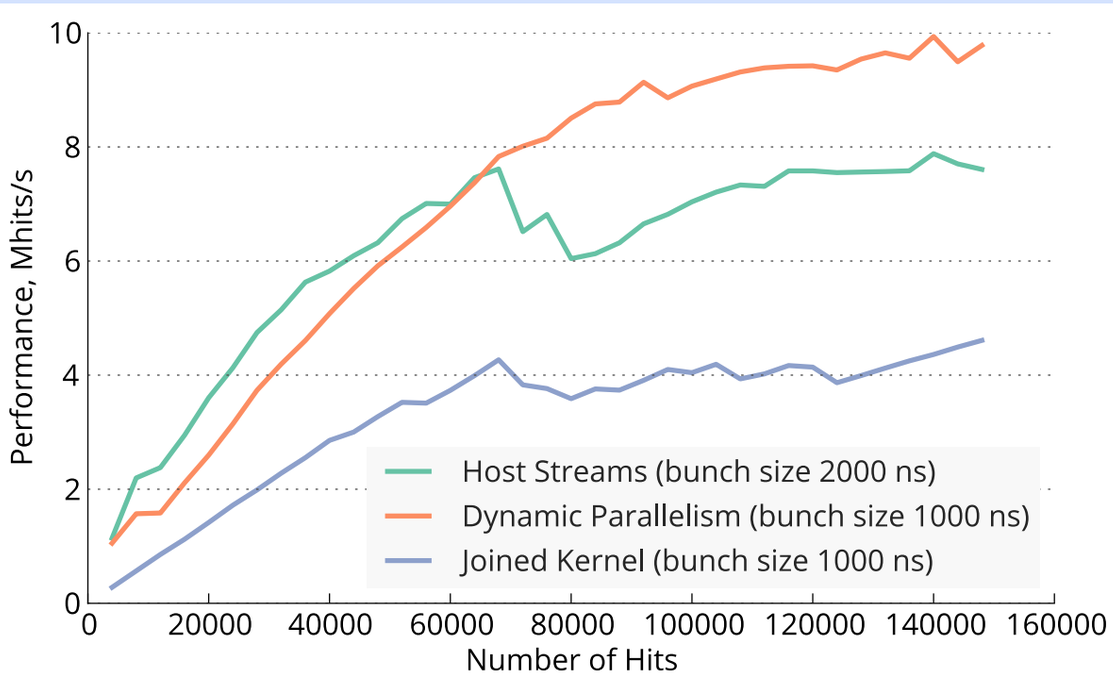
K20X

Kepler
Performance: 3.95 TFLOPS^{single}
Price: 3600 €

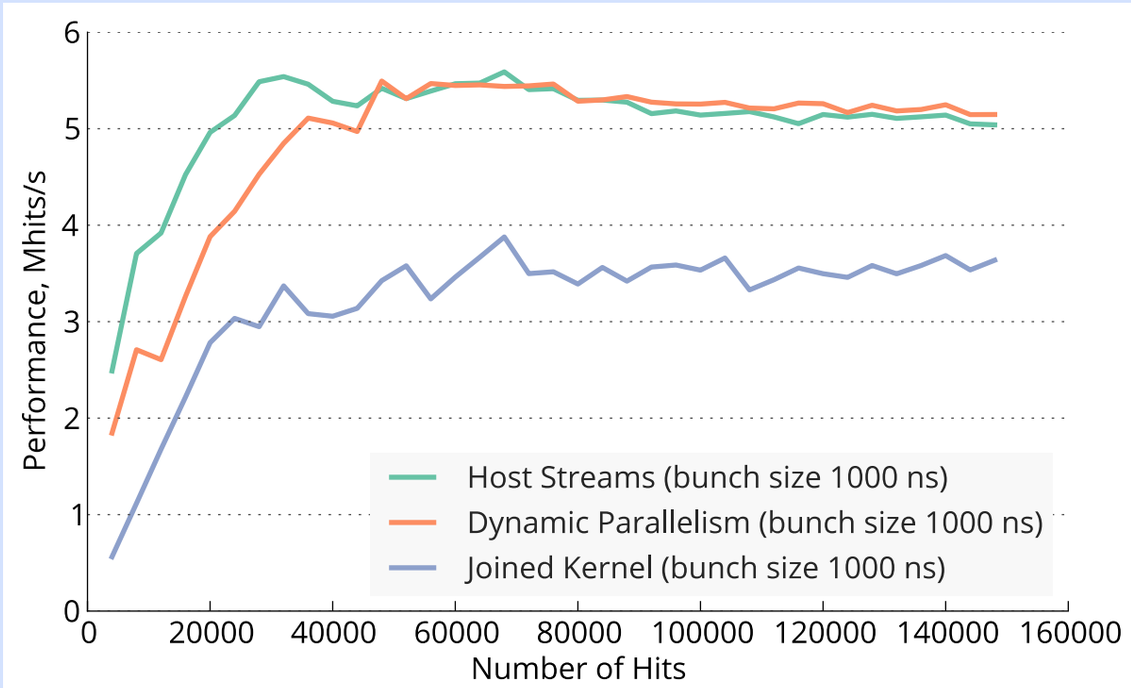
750 Ti

Maxwell
Performance: 1.3 TFLOPS^{single}
Price: 130 €

Triplet Finder — Kepler vs. Maxwell



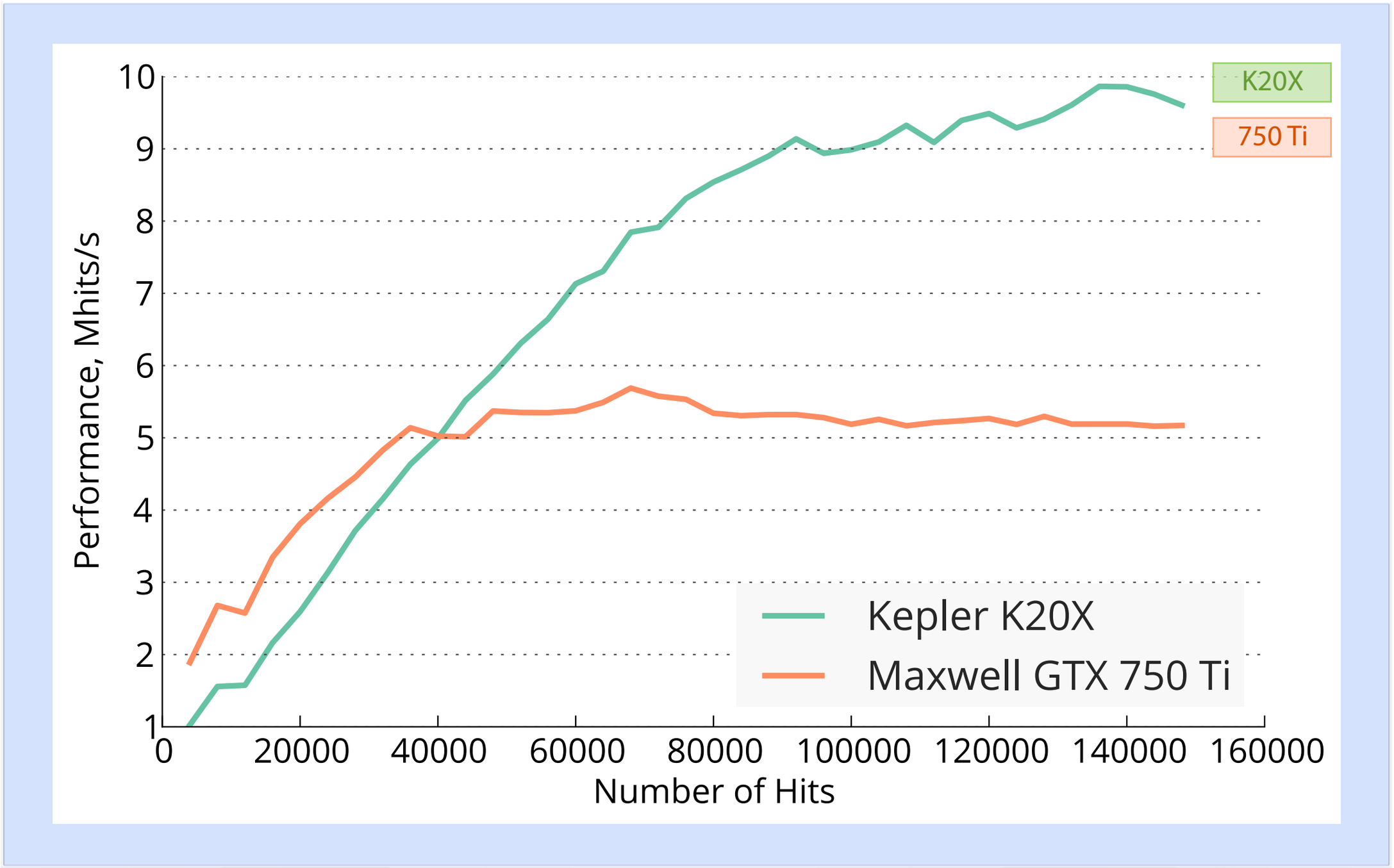
K20X



750 Ti

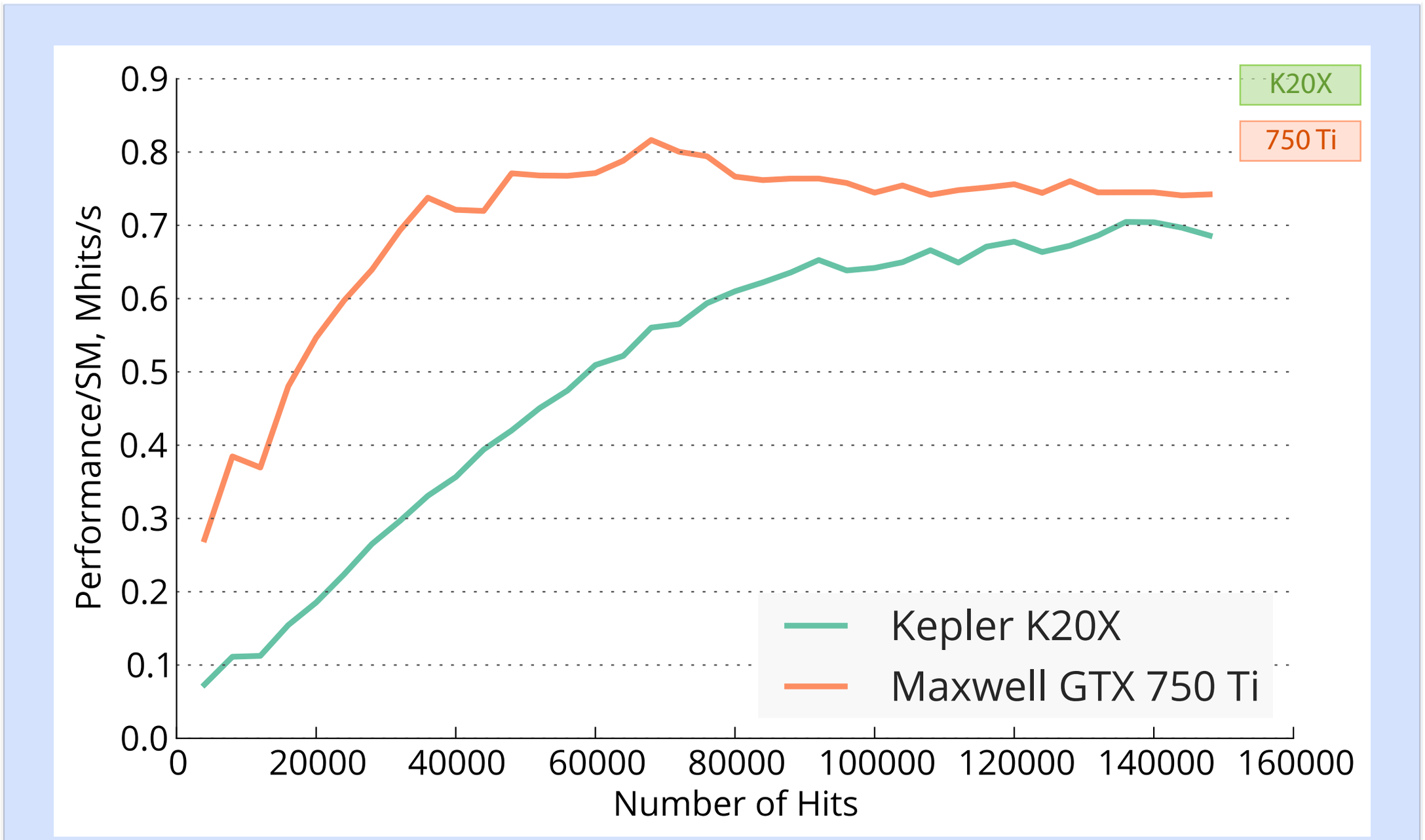
Triplet Finder — Kepler vs. Maxwell

Absolute Performance



Triplet Finder — Kepler vs. Maxwell

Performance per multiprocessor



Triplet Finder — Summary

- Best performance: **14 µs/event**
  - $14 \cdot 10^{-6}$ s/event * $2 \cdot 10^{7}$ event/s ⇒ 280 GPUs[2014]
  - PANDA[2019]: Multi GPU system – $\mathcal{O}(50)$ GPUs
- Still: Optimizations possible & needed
  - ε needs to be improved
  - Speed, €:
    - *More float less double*-cards a la K10

Summary

- Circle Hough: PandaRoot+GPU
- Triplet Finder: Max performance 12 GHit/s
- Riemann: Code ready to be put into PandaRoot
- Data transfer to GPU with FairMQ: Ludovico‘s talk

Summary

- Circle Hough: PandaRoot+GPU
- Triplet Finder: Max performance 12 GHit/s
- Riemann: Code ready to be put into PandaRoot
- Data transfer to GPU with FairMQ: Ludovico‘s talk

Thank you!

Andreas Herten
a.herten@fz-juelich.de

List of Resources Used

- **#11:** Flare Gun icon by Jop van der Kroef from The Noun Project
- **#12**: Flow Chart by Michael Wohlwend from The Noun Project
- **#27:** STT event animation by Marius C. Mertens
- **#35:** Graphics cards images by NVIDIA promotion
- **#35:** GPU Specifications
  - Tesla K20X Specifications: **http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf**
  - Tesla K40 Specifications: http://www.nvidia.com/content/PDF/kepler/Tesla-K40-Active-Board-Spec-BD-06949-001_v03.pdf
  - Tesla Familiy Overview: http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf

Backup

Triplet Finder — Optimizations

- Impact of chipset

	Tesla K40	Tesla K20X	GeForce GTX 760 Ti
Peak *double* performance	1.46 TFLOPS	1.31 TFLOPS	0.04 TFLOPS
Peak *single* performance	4.29 TFLOPS	3.95 TFLOPS	1.3 TFLOPS
GPU Chipset	GK110B	GK110	G**M**107
# CUDA Cores	2880	2688	640
Memory size	12 GB	6 GB	2 GB
Memory bandwidth	288 GByte/s	250 GByte/s	192 GByte/s

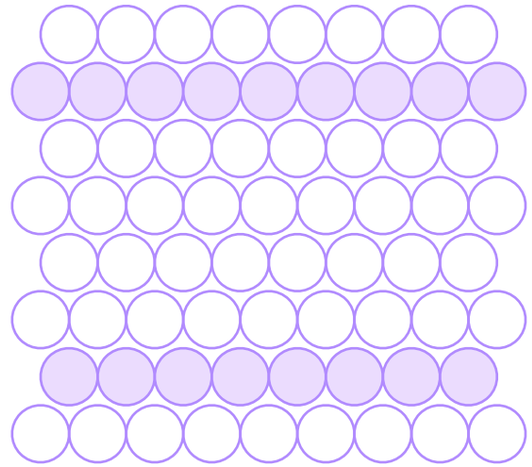
Source: http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf

Triplet Finder — Method

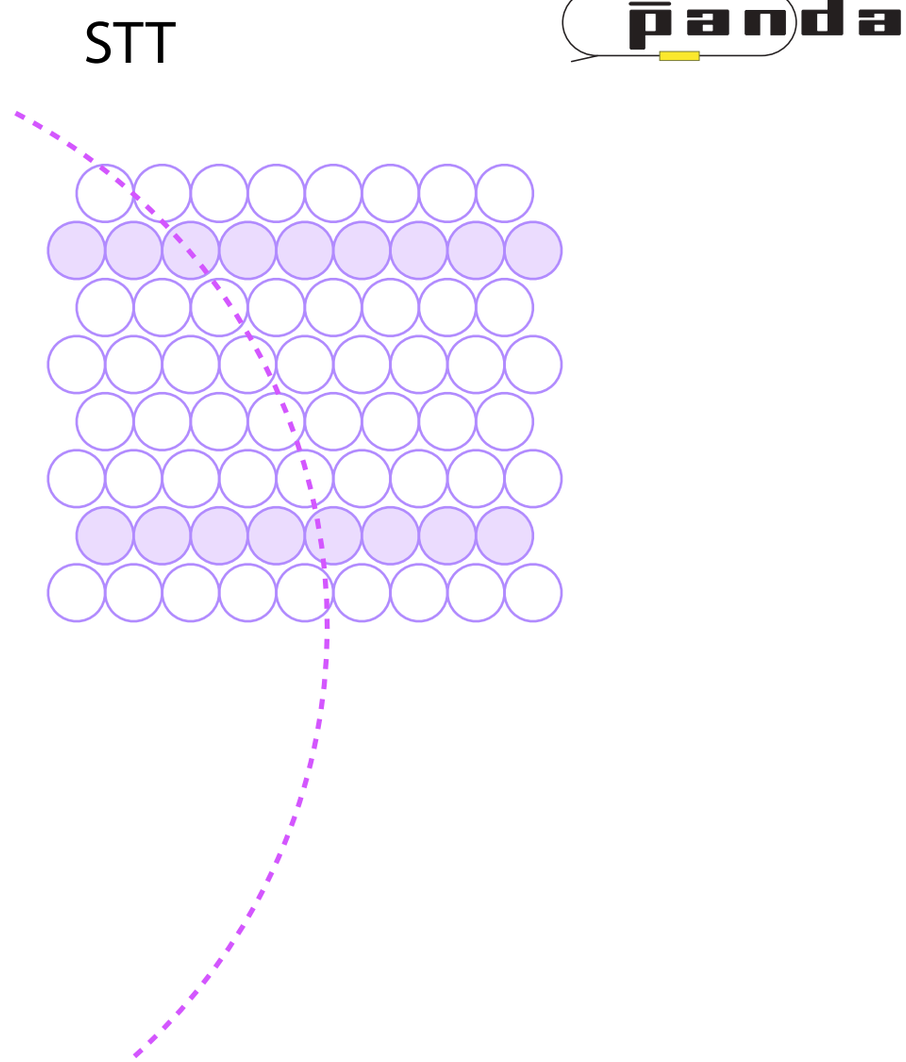
JÜLICH
FORSCHUNGSZENTRUM

panda

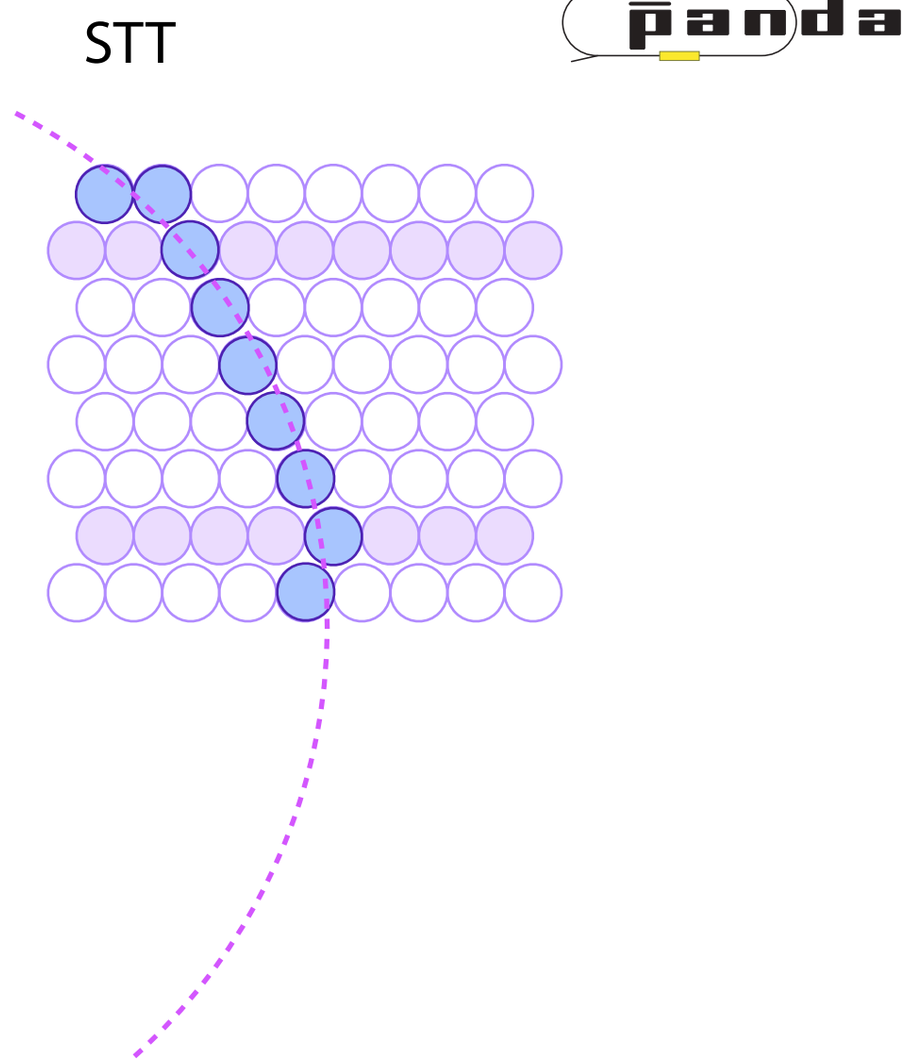
STT



Triplet Finder — Method

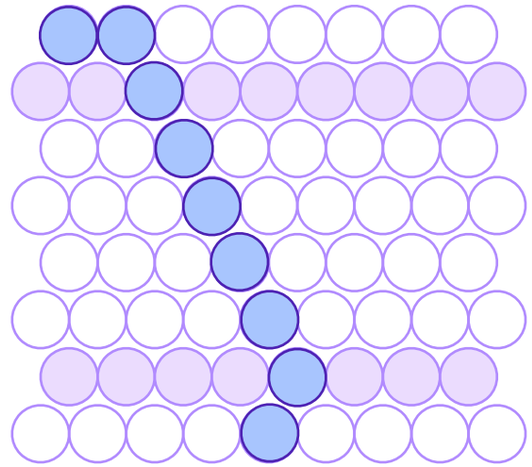


Triplet Finder — Method



Triplet Finder — Method

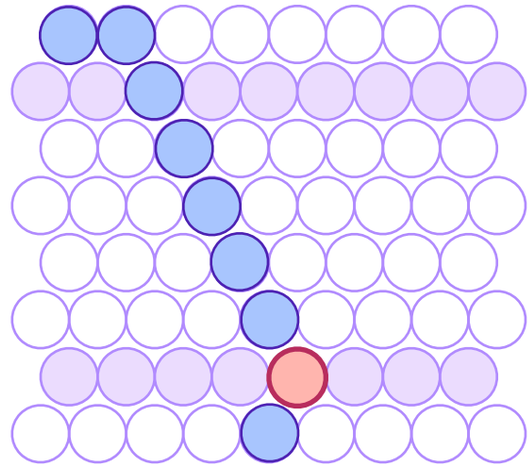
STT



Triplet Finder — Method

- STT hit in **pivot** straw

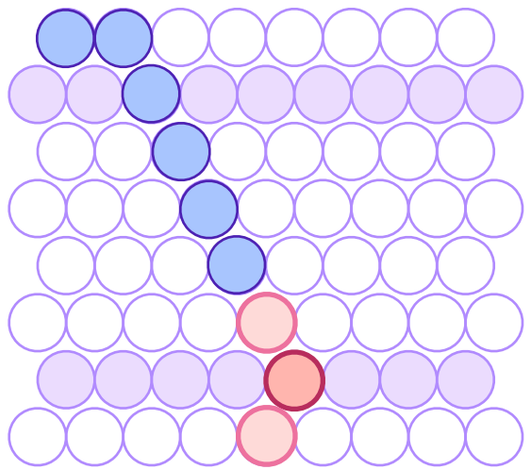
STT



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

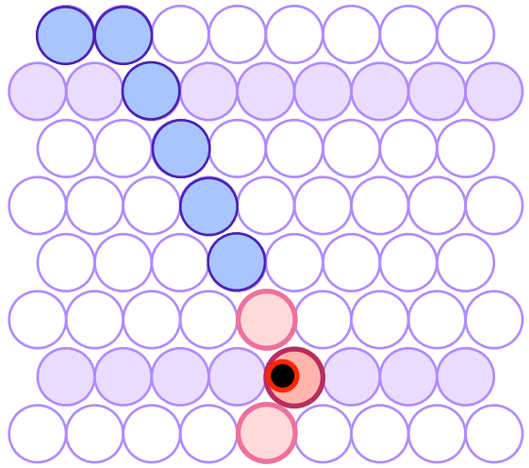
  → Create **virtual hit** *(triplet)* at center of gravity *(**cog**)*



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity *(**cog**)*
- **Combine** with

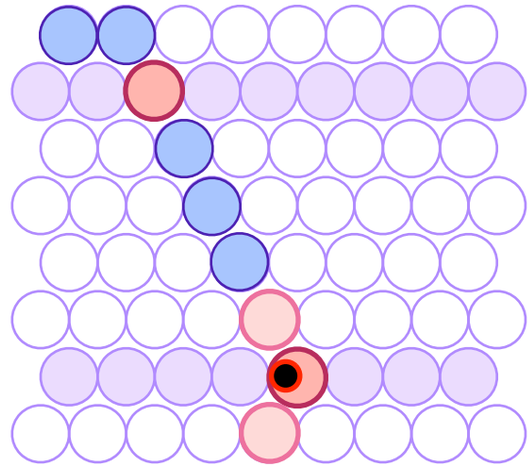


Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



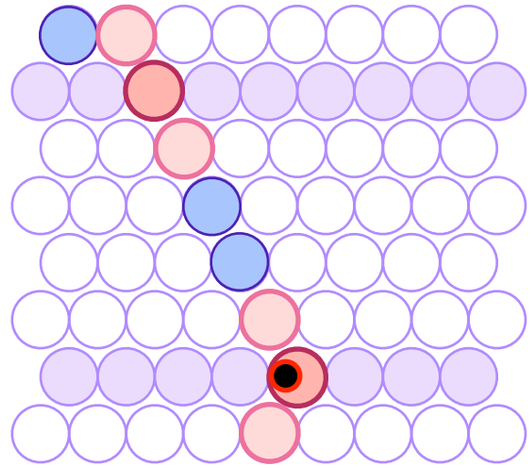
1. Second STT pivot-*cog* virtual hit

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



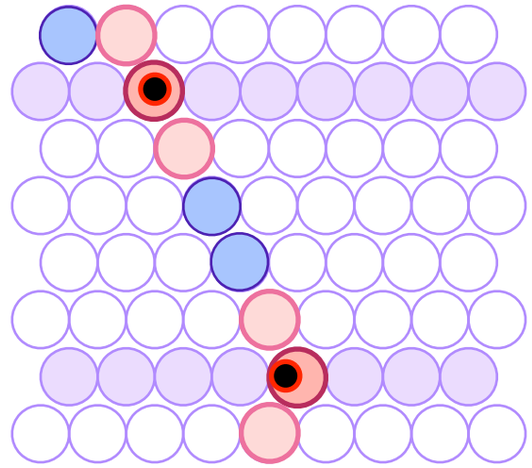
1. Second STT pivot-*cog* virtual hit

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)*
  at center of gravity ***(cog)***
- **Combine** with

STT



1. Second STT pivot-*cog* virtual hit

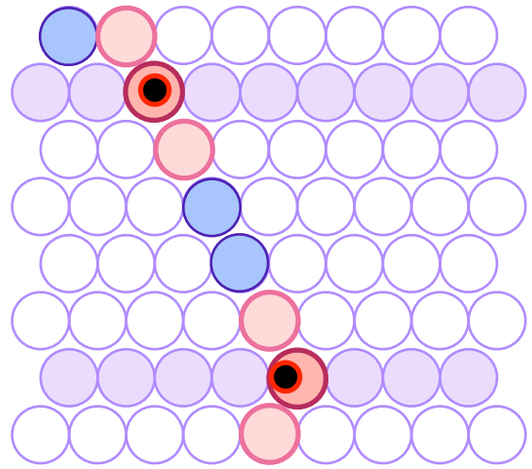
Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

1. Second STT pivot-*cog* virtual hit
2. Interaction point

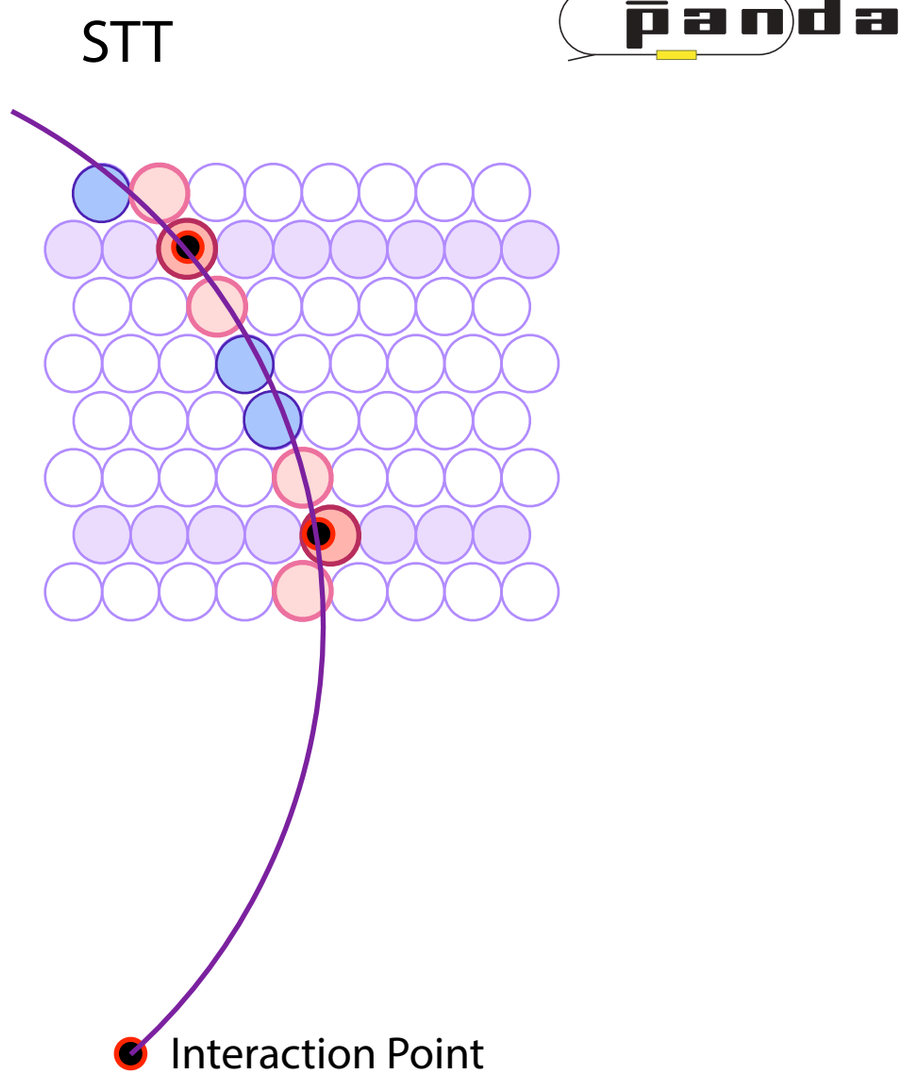
STT



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with
  1. Second STT pivot-*cog* virtual hit
  2. Interaction point
- Calculate **circle** through three points

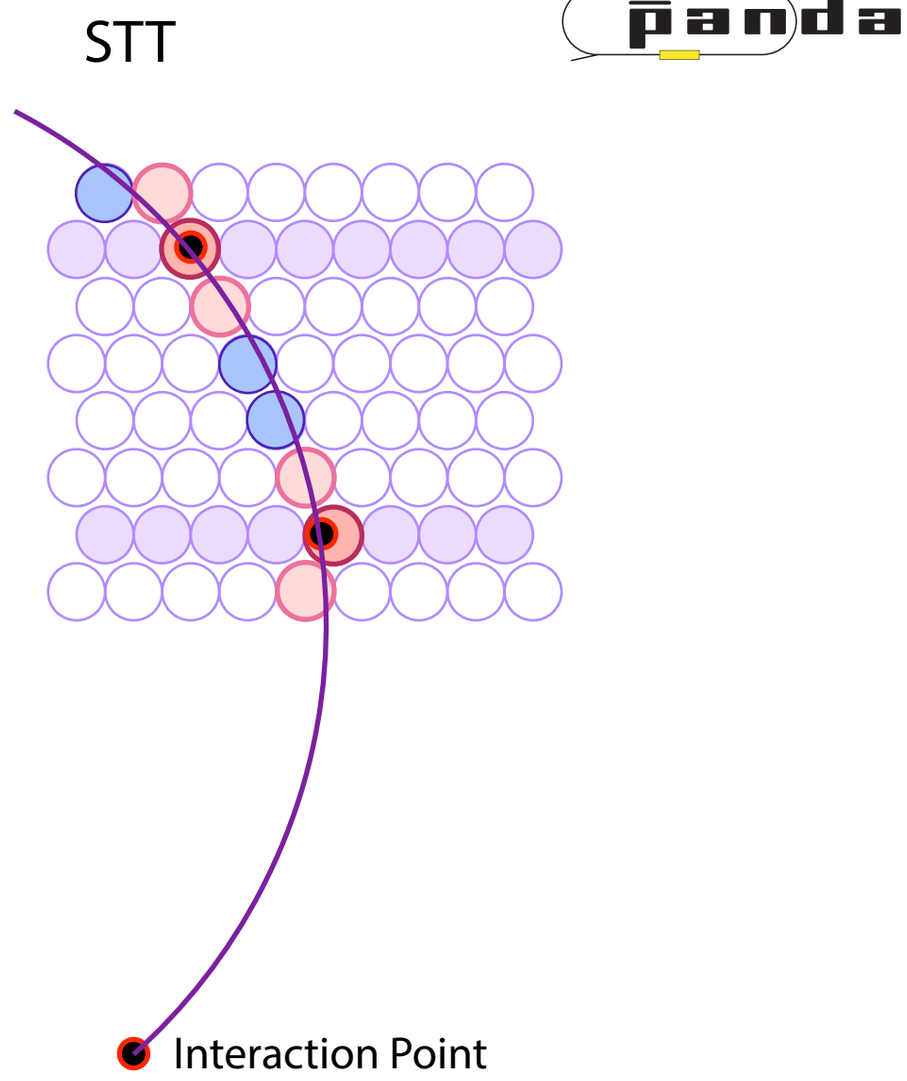


Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with
  1. Second STT pivot-*cog* virtual hit
  2. Interaction point
- Calculate **circle** through three points

→ **Track** Candidate



Triplet Finder — Kernel Launch Strategies

- Joined Kernel *(JK)*: slowest
  - High # registers → low occupancy
- Dynamic Parallelism *(DP)* / Host Streams *(HS)*: comparable performance
  - Performance
    - HS faster for small # processed hits, DP faster for > 45000 hits
    - HS stagnates there, while DP continues rising
  - Limiting factor
    - High # of required kernel calls
    - Kernel launch latency
    - Memcopy
  - HS more affected by this, because
    - More PCI-E transfers (launch configurations for kernels)
    - Less launch throughput, kernel launch latency gets more important
    - False dependencies of launched kernels
      - Single CPU thread handles all CUDA streams (Multi-thread possible, but synchronization overhead too high for good performance)
      - Grid scheduling done on hardware (Grid Management Unit) (DP: software)
        - » False dependencies when N(streams) > N(device connections)=32[3.5]

Triplet Finder — Host Stream Connections