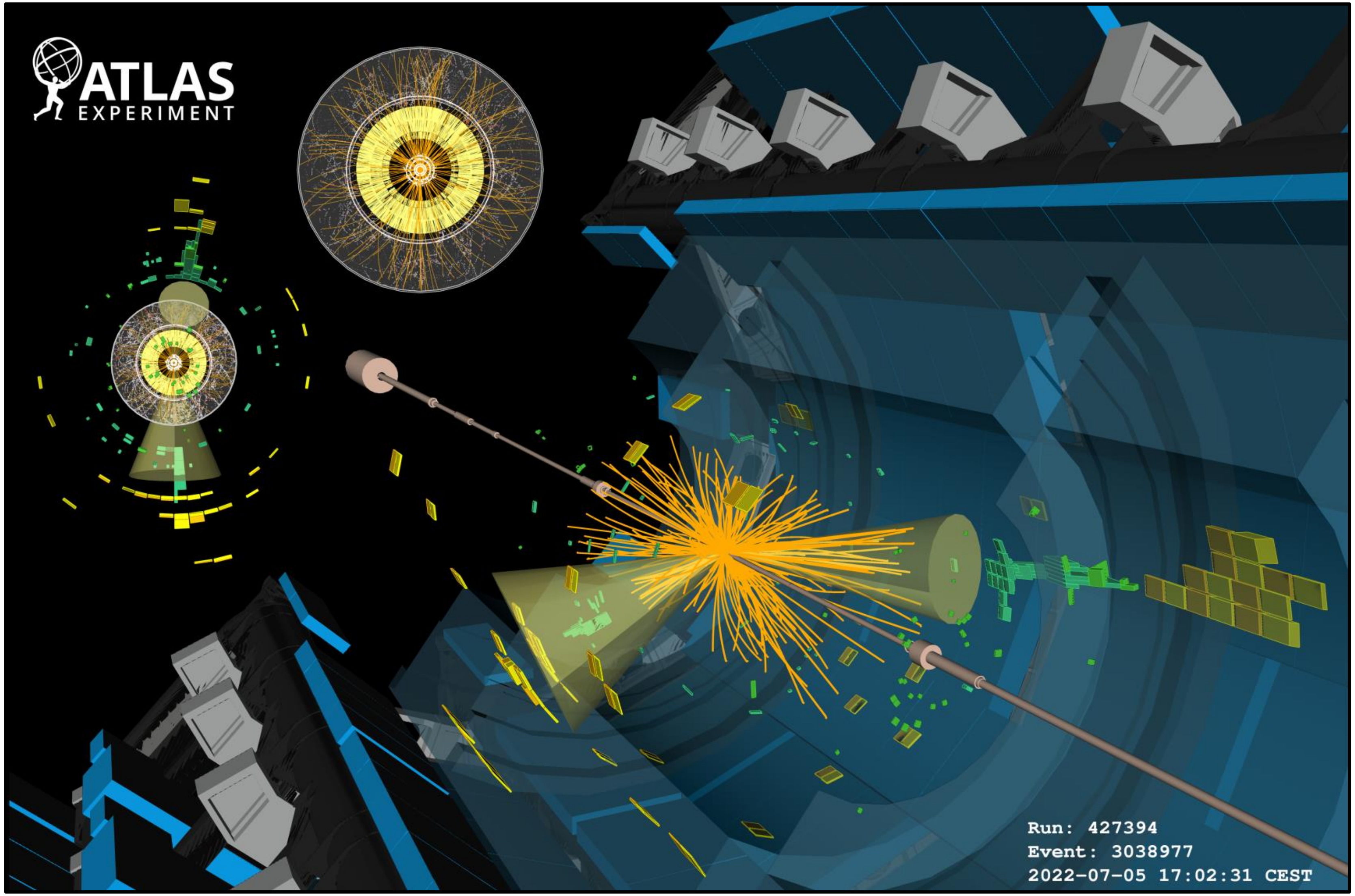
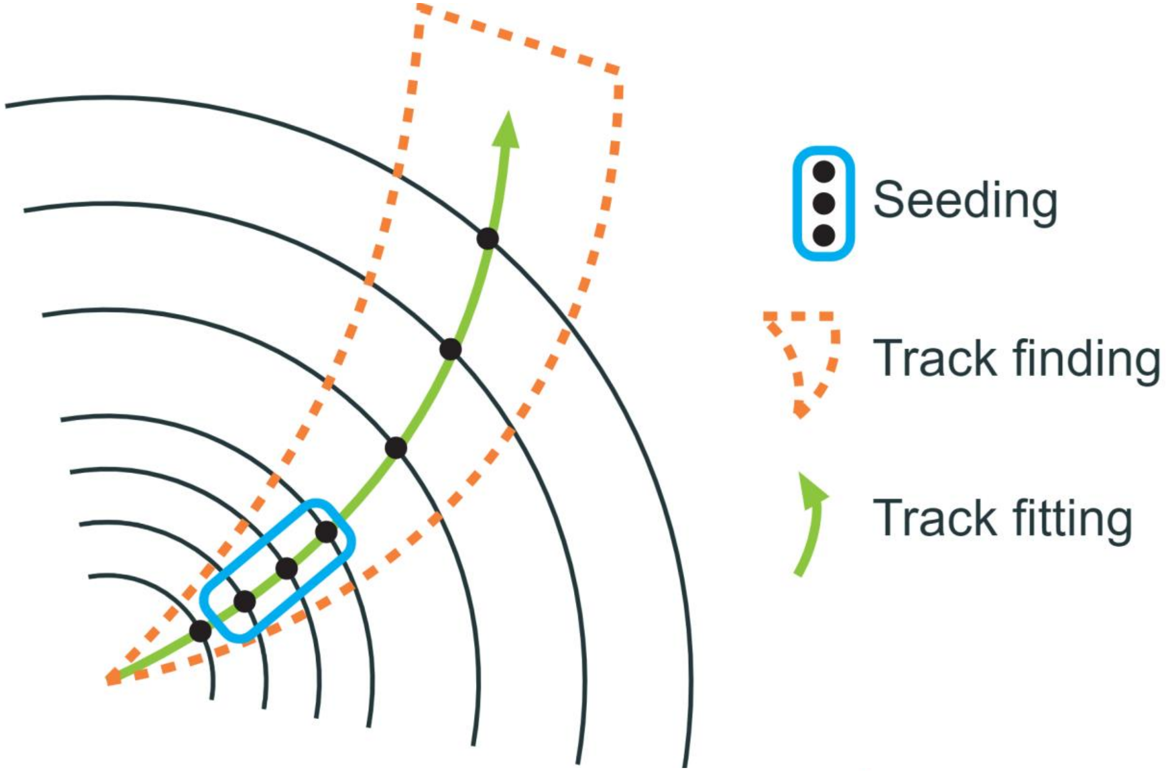
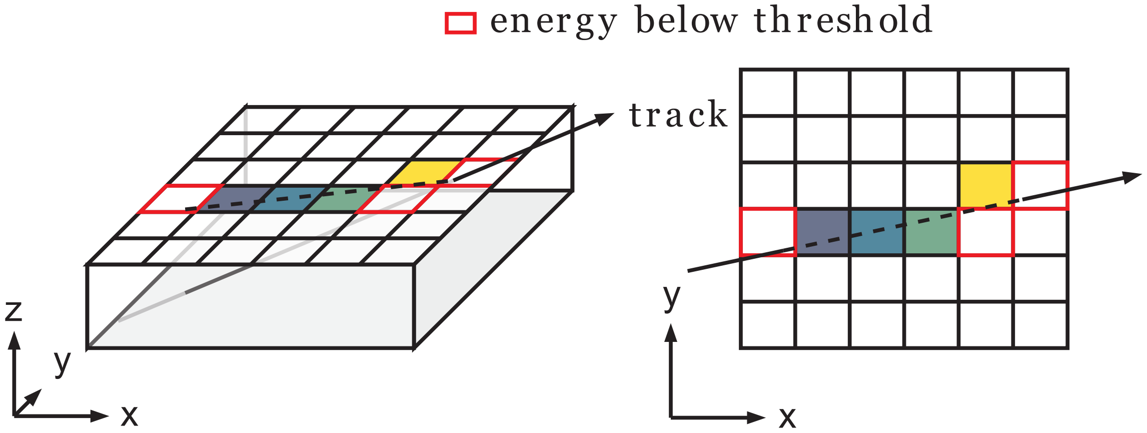
Accelerating track reconstruction algorithms using GPUs at ATLAS

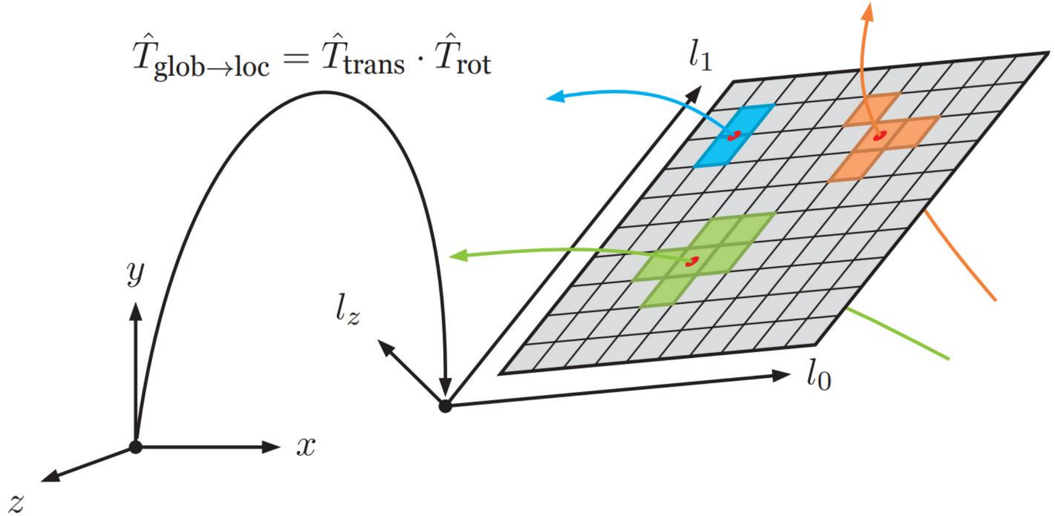


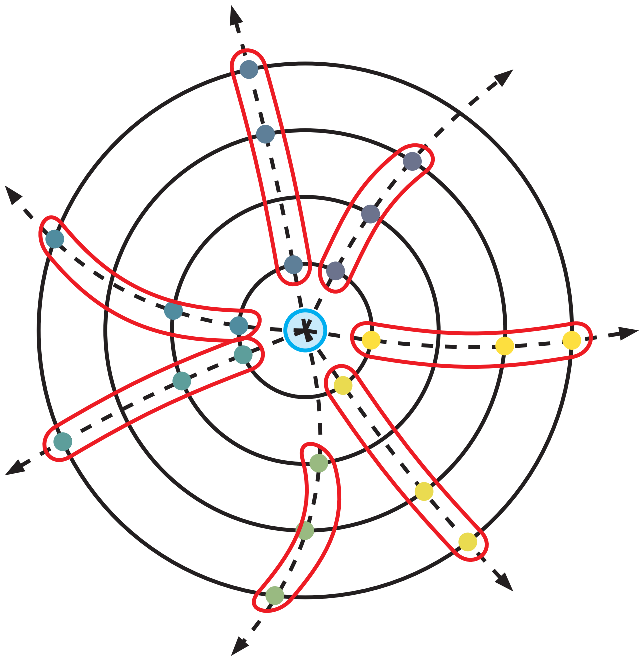
Guilherme Almeida CERN | LIP 2023

ACTS track Reconstruction



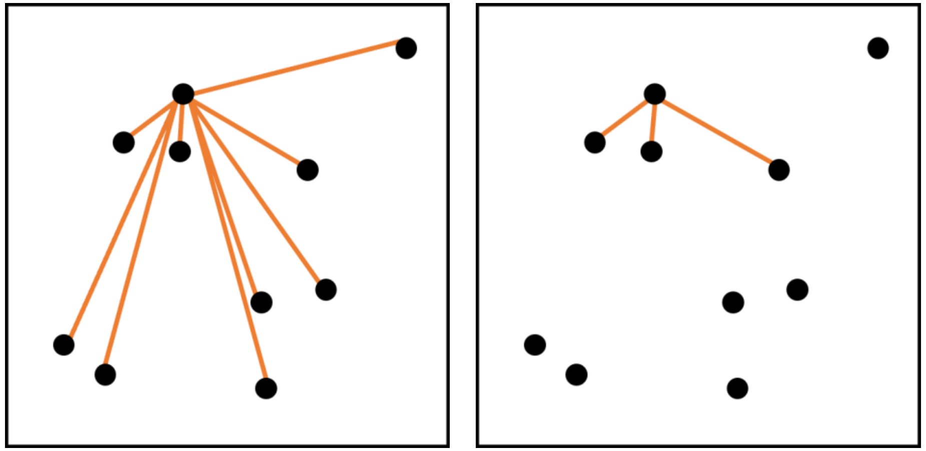


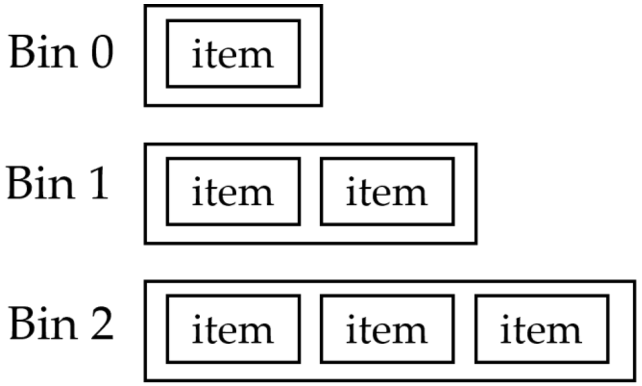

$\hat{T}_{\text{glob}\rightarrow\text{loc}} = \hat{T}_{\text{trans}} \cdot \hat{T}_{\text{rot}}$
l_1
l_z
l_0
y
x
z



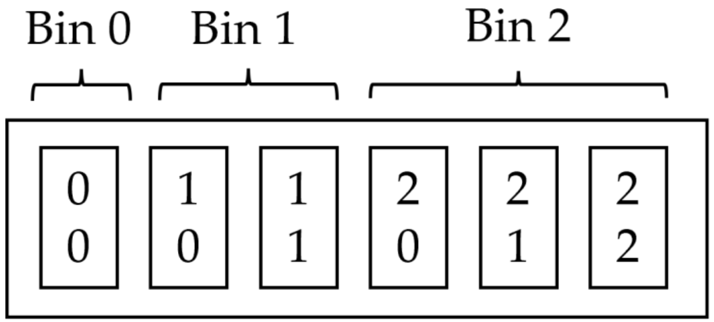
traccc project

- ACTS track reconstruction on GPU
- No physics performance loss
- Massively parallel programming
- No dynamic memory allocation in GPU threads
- Count/Find kernels rather than overallocation
- Prefix sum vector for linking a thread number to a jagged position

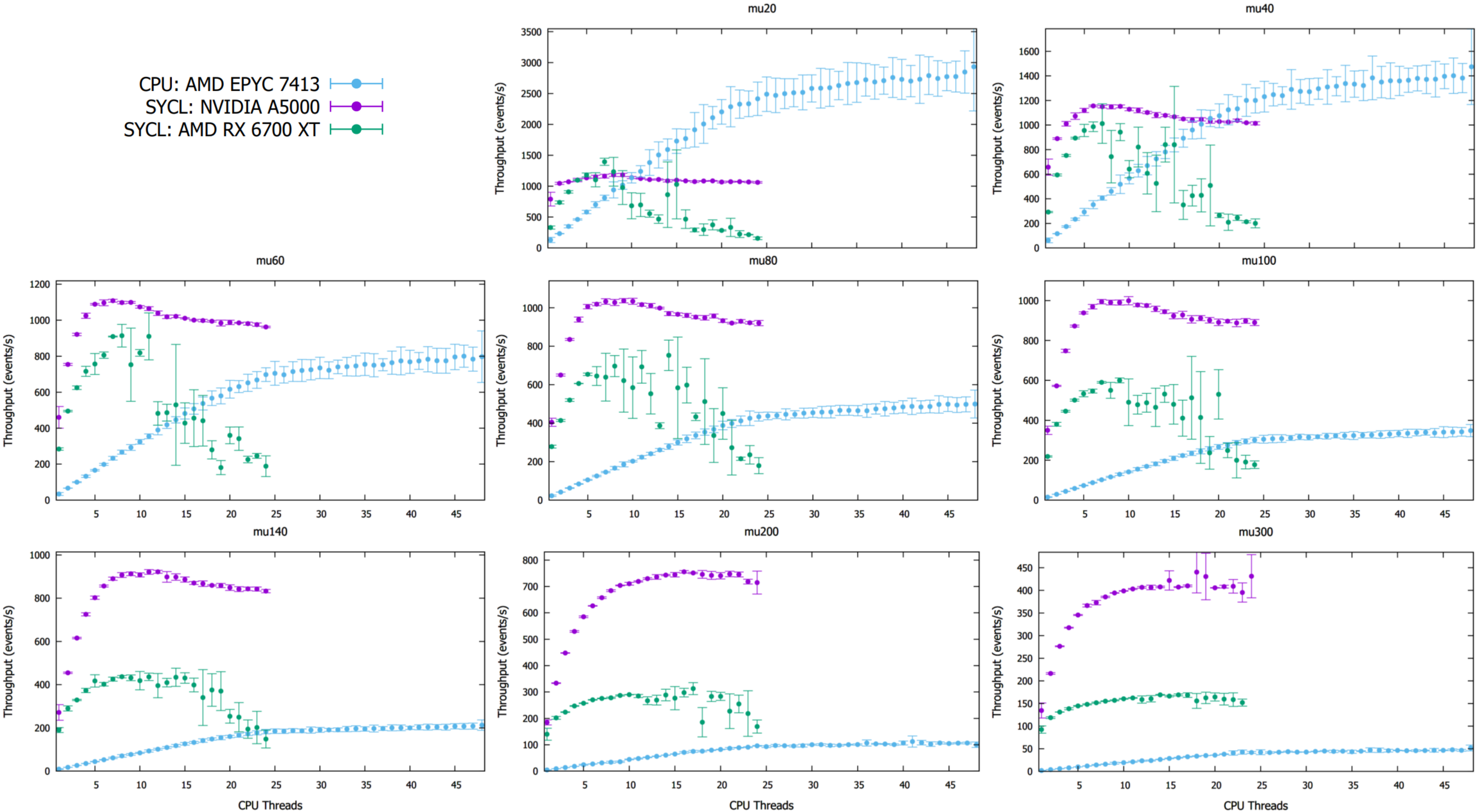




Input jagged vector

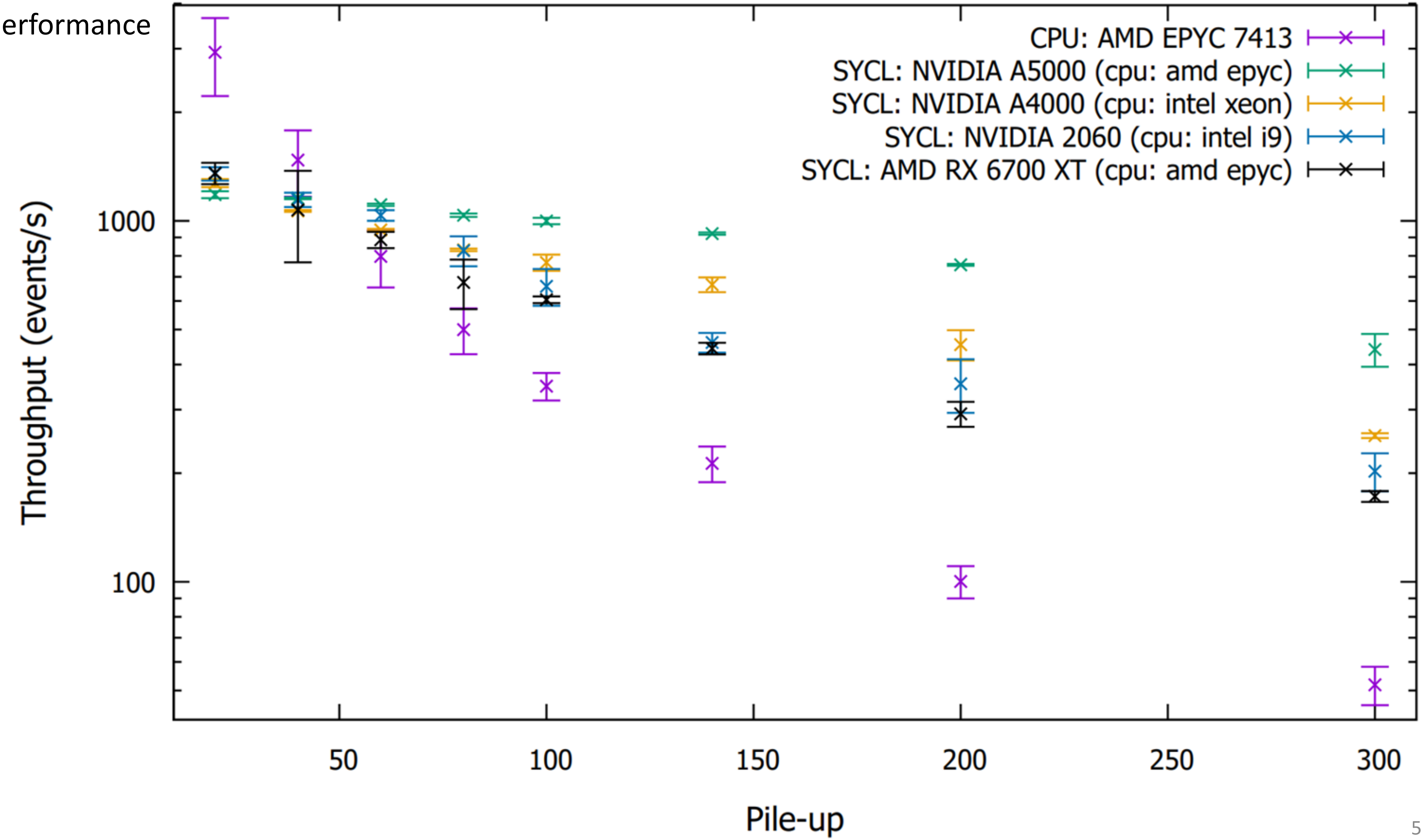


Auxiliary prefix sum vector



CPU: AMD EPYC 7413
SYCL: NVIDIA A5000
SYCL: AMD RX 6700 XT
mu20
mu40
mu60
mu80
mu100
mu140
mu200
mu300
Throughput (events/s)
CPU Threads
0
500
1000
1500
2000
2500
3000
3500
200
400
600
800
1000
1200
1400
1600
5
10
15
20
25
30
35
40
45
50
100
150
250
300
350
450
700

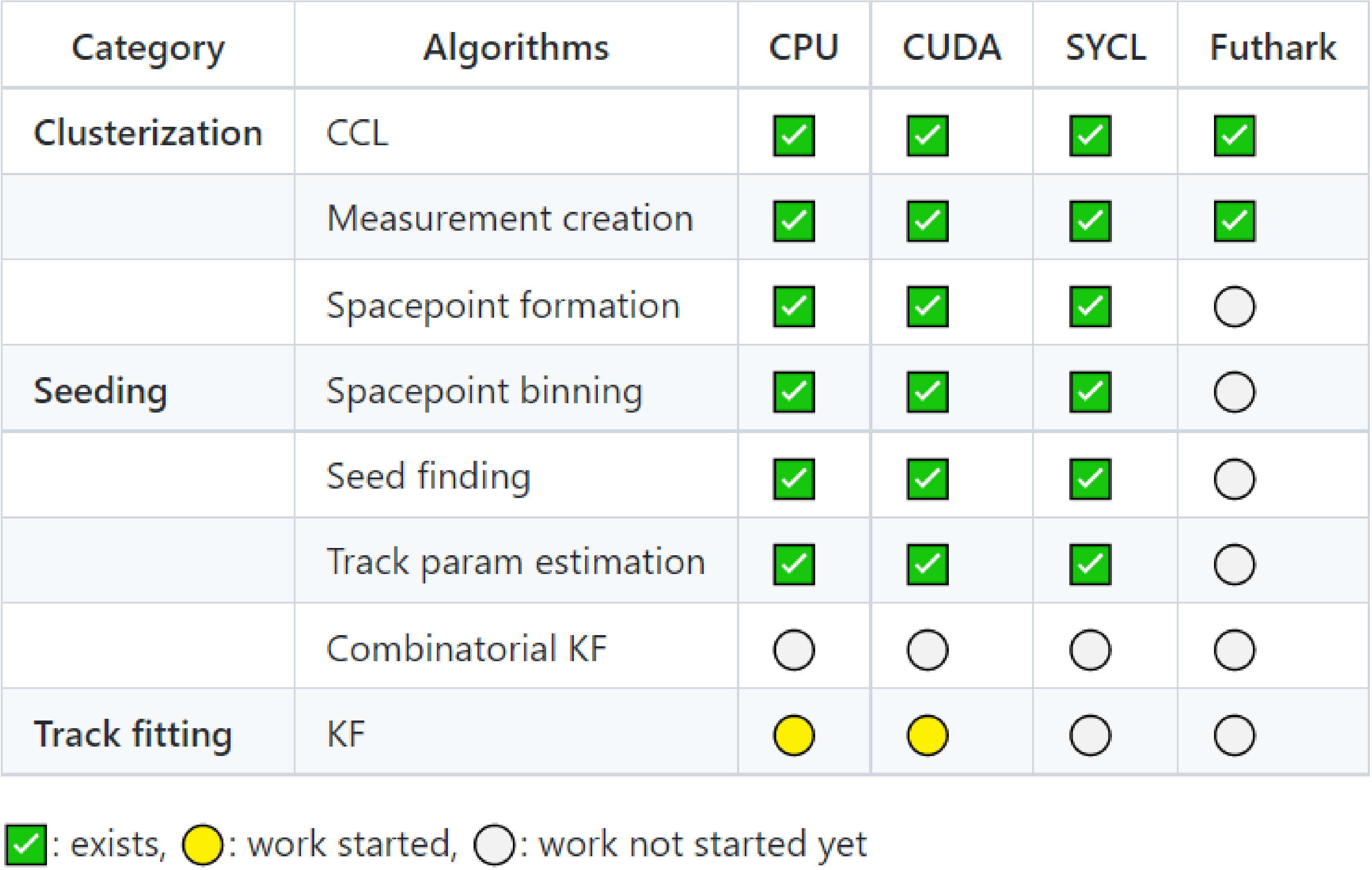
Measurements at peak performance



Backup

Current status - functionality

- Different backends
  - focus on SYCL/CUDA
- Large part of track reconstruction running entirely on the GPU
- Still being extended



Category	Algorithms	CPU	CUDA	SYCL	Futhark
Clusterization	CCL	✅	✅	✅	✅
	Measurement creation	✅	✅	✅	✅
	Spacepoint formation	✅	✅	✅	⚪
Seeding	Spacepoint binning	✅	✅	✅	⚪
	Seed finding	✅	✅	✅	⚪
	Track param estimation	✅	✅	✅	⚪
	Combinatorial KF	⚪	⚪	⚪	⚪
Track fitting	KF	🟡	🟡	⚪	⚪

✅: exists, 🟡: work started, ⚪: work not started yet

CPU vs GPU algorithm

- Running generic algorithm with input jagged vector and output jagged vector

on a CPU:

- Do calculations on input
- Append new members to result

on a GPU:

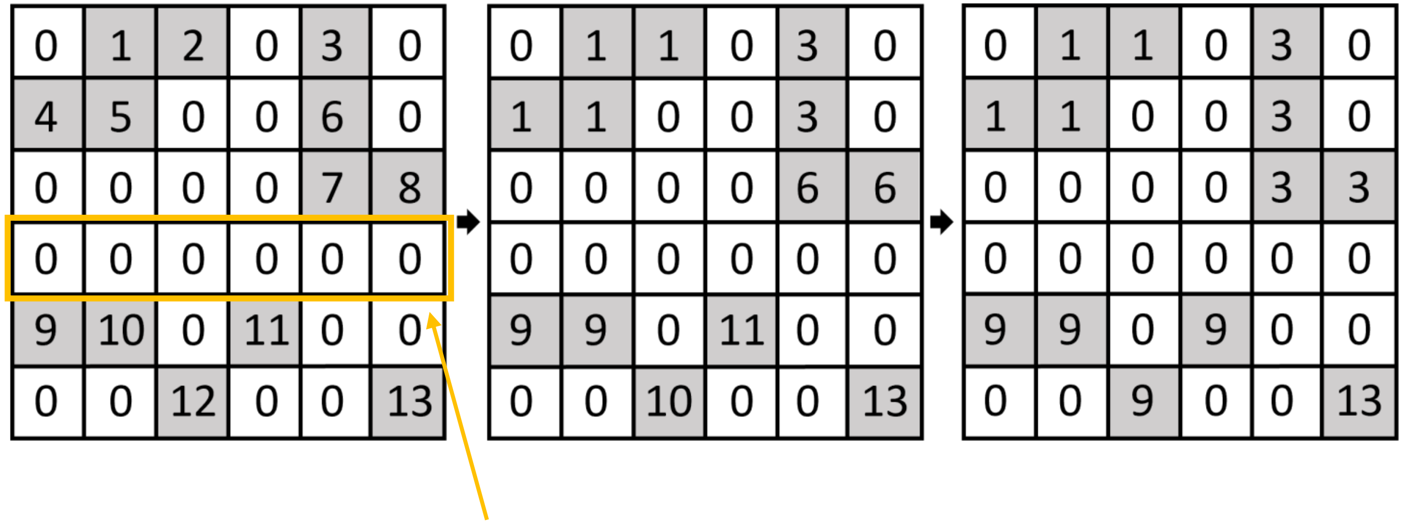
- 1st kernel – Create prefix sum vector for input jagged vector
- 2nd kernel – Count number of members needed for result
- 3rd kernel – Create prefix sum vector for counted items
- 4th kernel – Fill result jagged vector

+ CPU memory allocation between kernels

- Code itself can be very different to boost GPU resource usage
- GPU introduces a lot of complexity
- Throughput gain from massive parallelism

New clustering

- From module to cell parallelism
- No jagged data in clustering
- Organise data in equally sized partitions (on CPU)

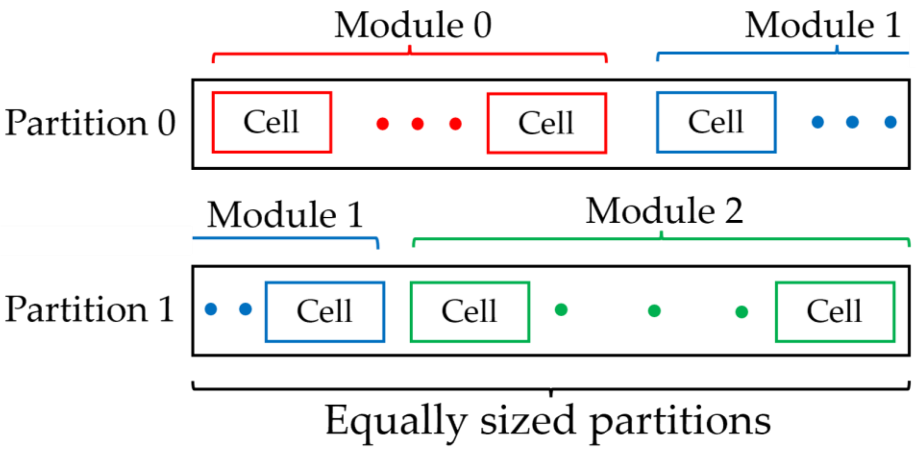


0	1	2	0	3	0
4	5	0	0	6	0
0	0	0	0	7	8
0	0	0	0	0	0
9	10	0	11	0	0
0	0	12	0	0	13

0	1	1	0	3	0
1	1	0	0	3	0
0	0	0	0	6	6
0	0	0	0	0	0
9	9	0	11	0	0
0	0	10	0	0	13

0	1	1	0	3	0
1	1	0	0	3	0
0	0	0	0	3	3
0	0	0	0	0	0
9	9	0	9	0	0
0	0	9	0	0	13

Possible partitioning point

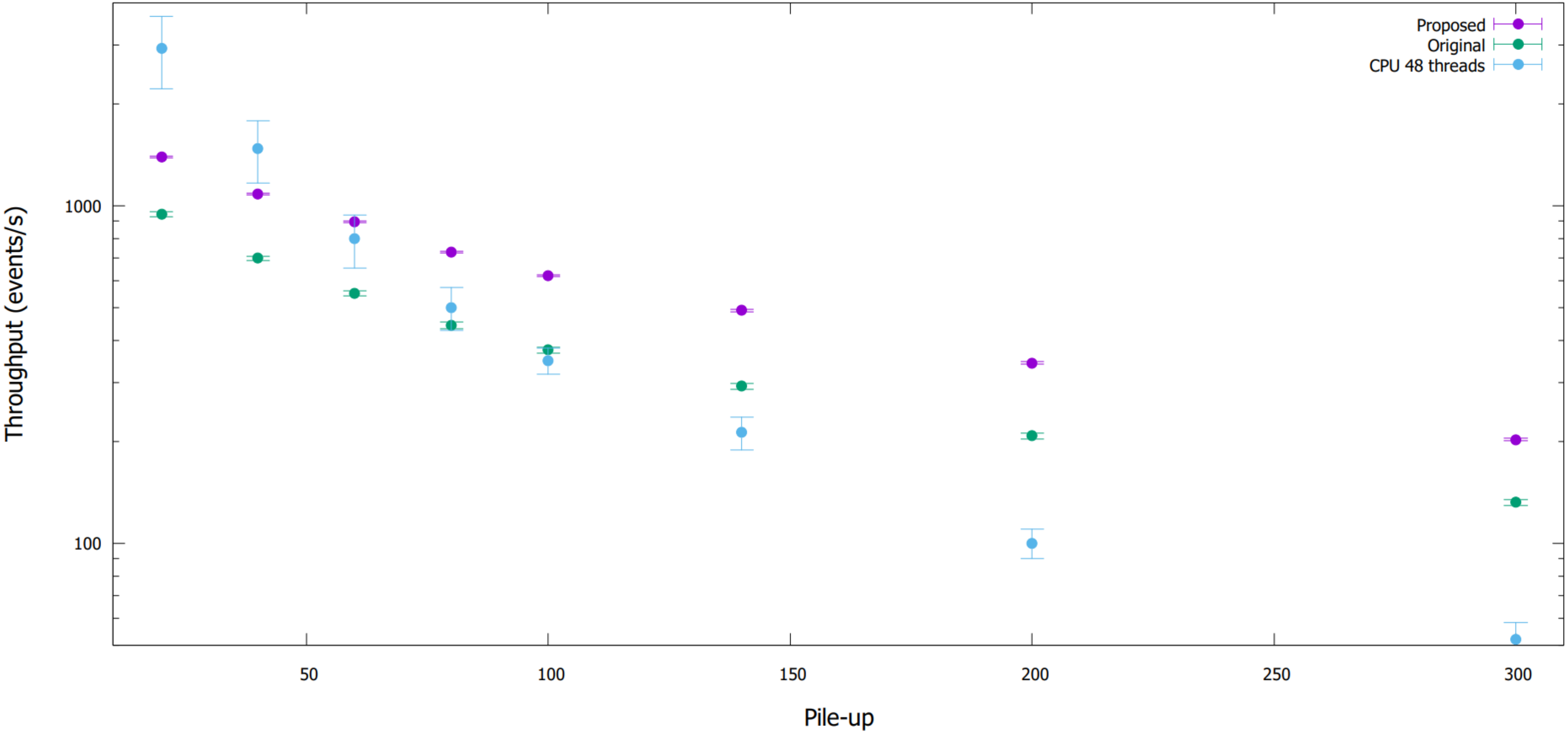


Equally sized partitions

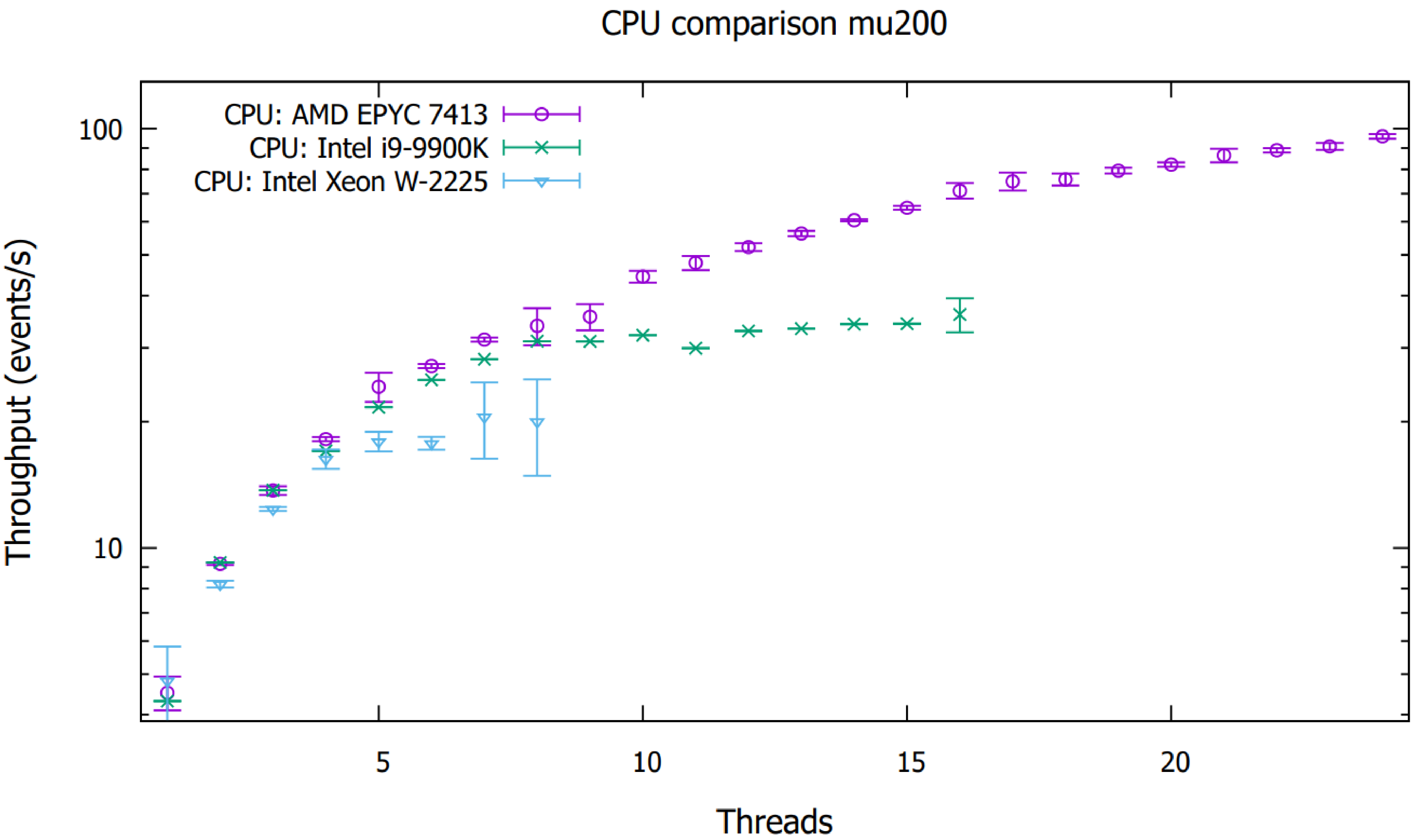
For the coding details, see the open PR in GitHub

New clustering - performance

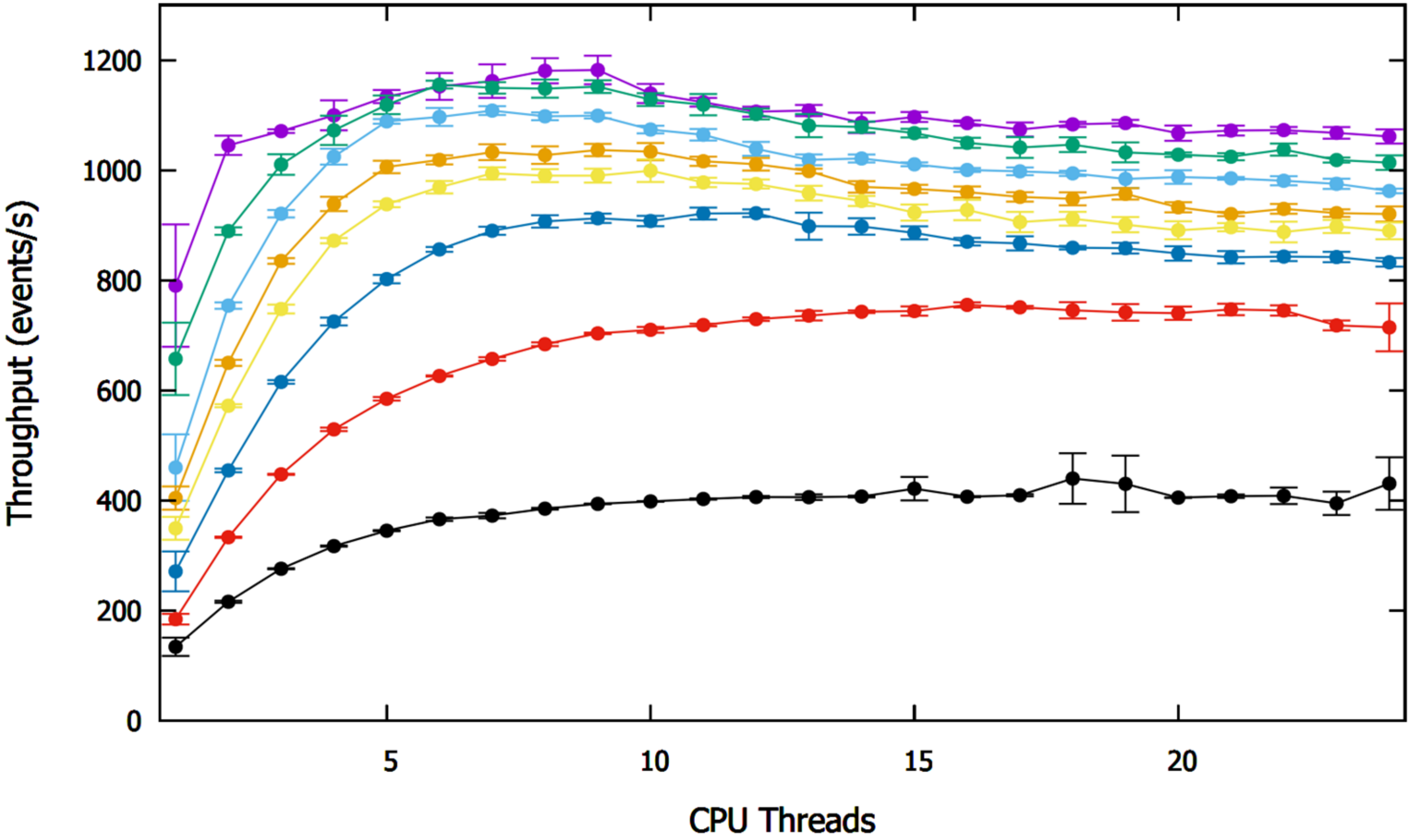
CPU: AMD EPYC 7413
GPU: NVIDIA RTX A5000



- WIP: 50% – 70% throughput improvement for full application


CPU comparison mu200
CPU: AMD EPYC 7413
CPU: Intel i9-9900K
CPU: Intel Xeon W-2225
Throughput (events/s)
100
10
5
10
15
20
Threads

SYCL: NVIDIA A5000 (cpu: amd epyc)
at different pile-up events


Throughput (events/s)
CPU Threads