Traccc as-a-service development update

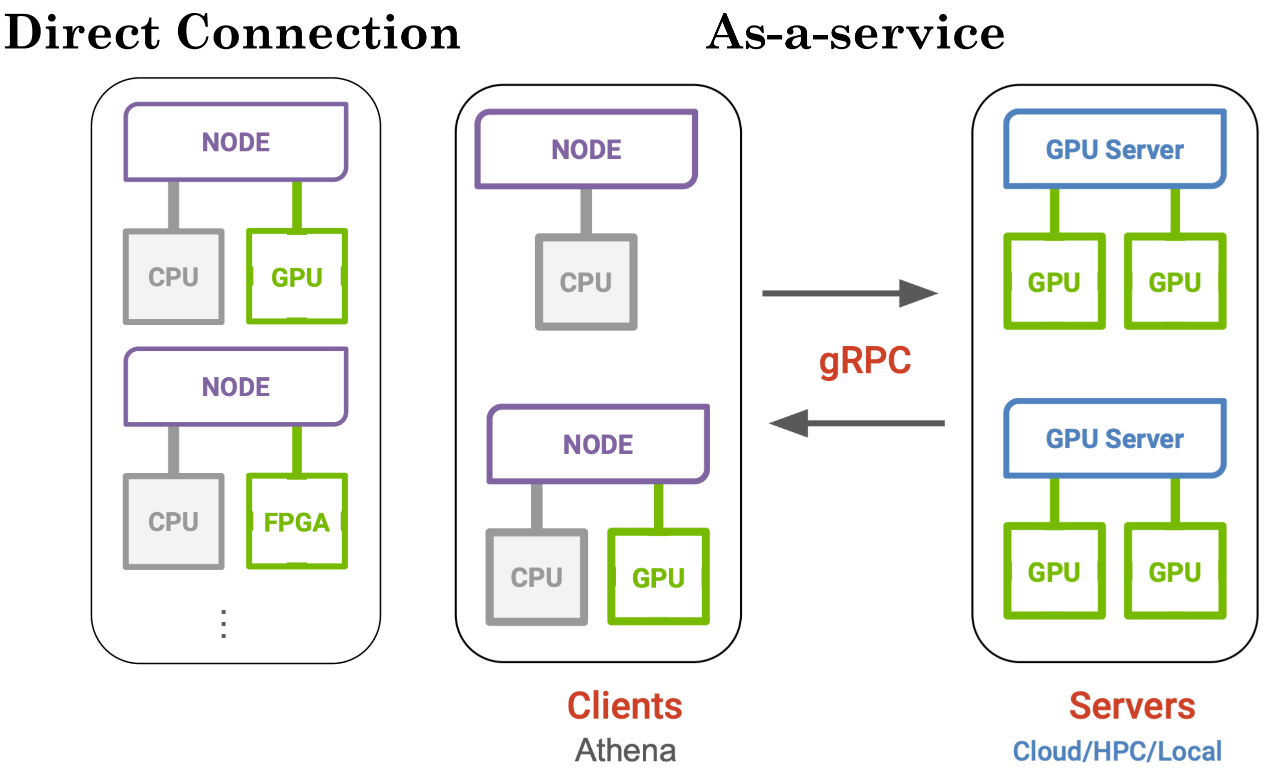
Miles Cochran-Branson, Yuan-Tang Chou, Xiangyang Ju, Haoran Zhao

ACTS parallelization meeting

6 September 2024

As-a-service (aaS)

- Heterogenous computing
  - CPU / GPU connected on the same node
  - Many working examples available
  - **Can be inefficient in use of resources**
- As-a-service model
  - Dedicated GPU server to offload computation
  - Can be easier to integrate with production framework (e.g. Athena)
  - **Potentially improve scalability and resource utilization**

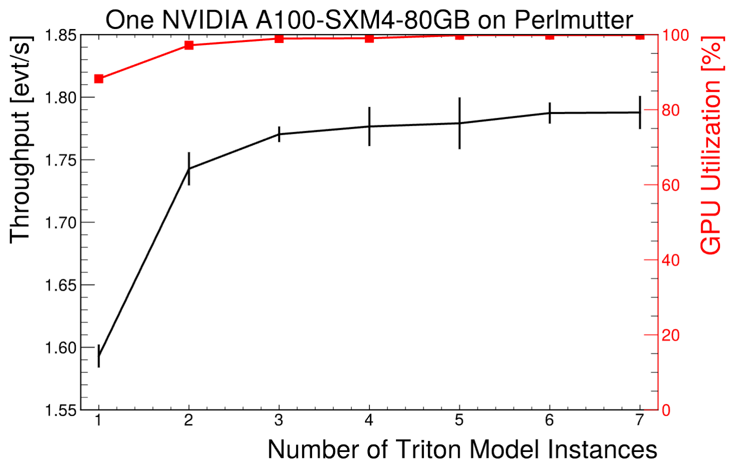


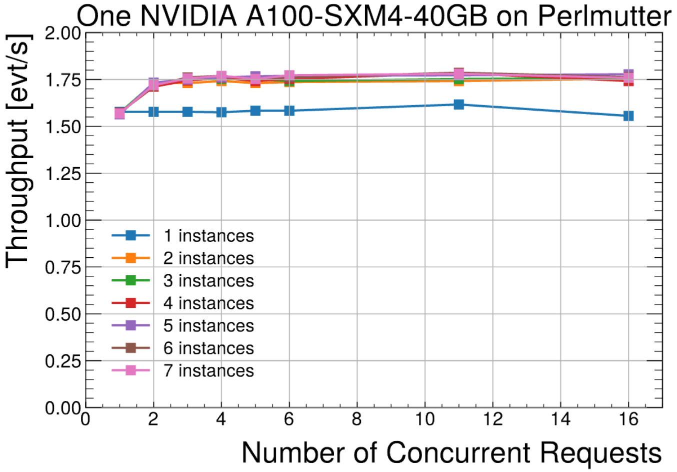
Yuan-Tang Chou, Inner Detector Tracking Workshop, 2024

Previous use: ExaTrkX-as-a-service

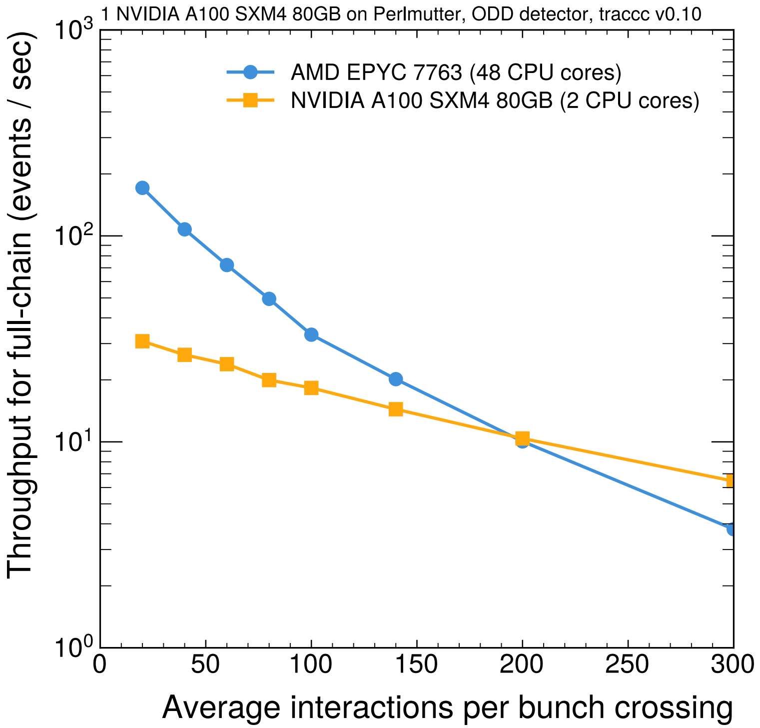
- With multiple model instances on the server, GPU utilization increases to ~100%
  - Low overhead of server ⇒ one instance ~standalone performance
- With multiple concurrent requests, throughput increases
  - Steady around 2-3 concurrent requests
- Demonstrates usefulness of aaS approach

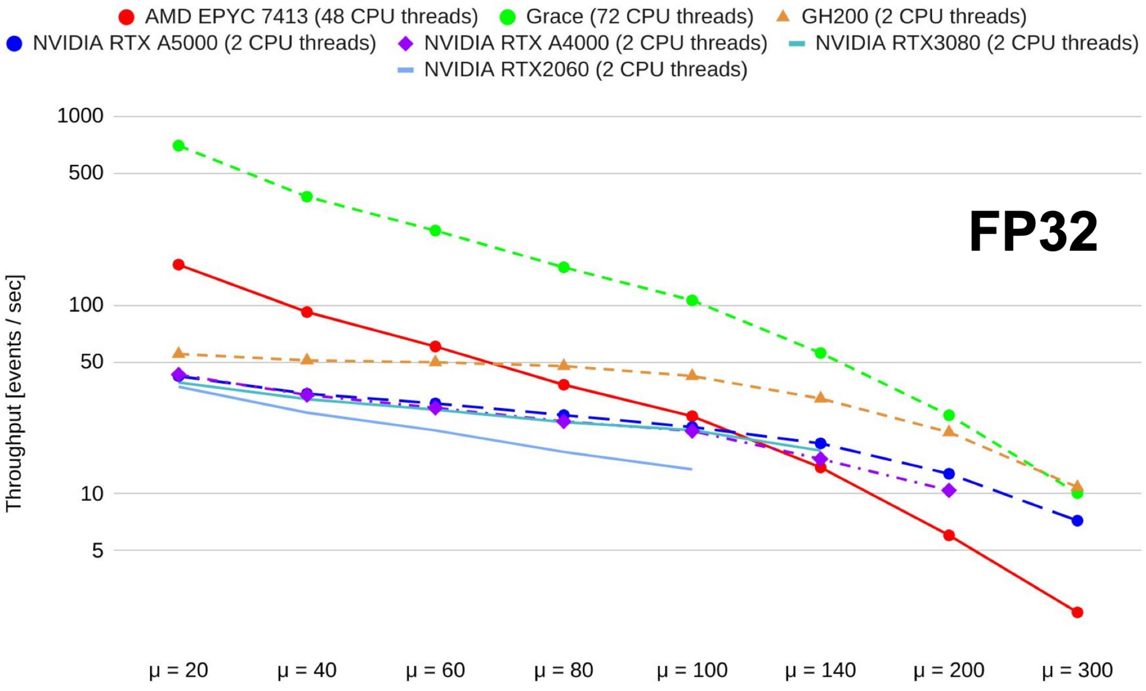
ODD detector with $\mu = 200$





Traccc standalone performance



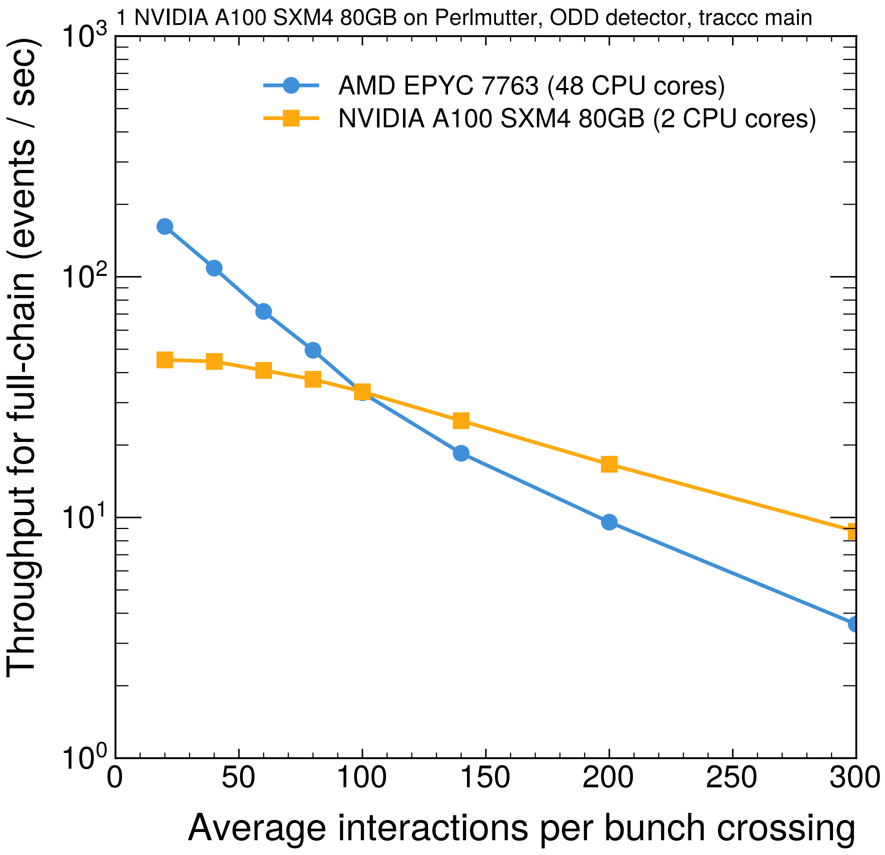


[Attila Krasznahorkay, Inner Detector Tracking Workshop, 2024]

For traccc v0.10, able to match performance presented by Attilia at inner detector workshop on A100 at Perlmutter

Tracc main performance

- Previously showed poor performance of main traccc
- Now see improvement in GPU performance
- Following results are still for v0.10
  - Dealing with silly bugs migrating to main version



Traccc as-a-Service implementation

- Main components:
  - Standalone version of traccc to run
  - Backend to execute standalone version on server
  - Client to send and receive data from server
- Server is simply an interactive node on Perlmutter
  - Send and receive data over localhost

Standalone

- `initialize()`
  - Read detector, geometry, and digitization files
  - Setup detector, magnetic field
  - Copy detector to device memory
  - Configure finding and fitting options
- `run(std::vector<traccc::io::csv::cell> cells)`
  - Read cells into device memory
  - Perform algorithm (clusterization, spacepoint formation, track finding, and track fitting)

Custom Backend

- Built using NVIDIA Triton server

1. Initialization
   a. Initialize server
   b. Run initialize function from standalone
2. Run
   a. Process tensor of cell components from client and embed in detector
   b. Convert to `std::vector<traccc::io::csv::cell> cells`
   c. Run pipeline from standalone

- **To test standalone performance, 2a and 2b are instead done in initialization**
  - Will eventually be done on the client side and sent over via direct memory buffer

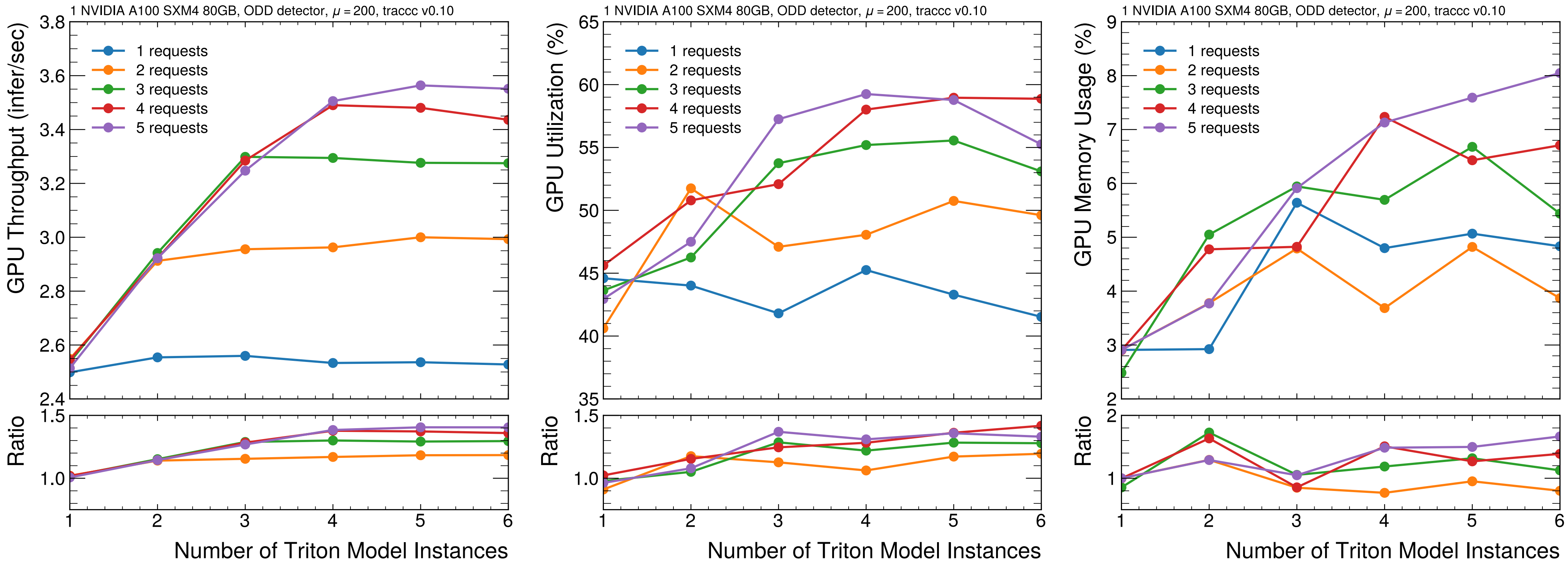
Client

- Reads .csv containing cell information
- Send data to server for processing
- **For standalone test, sending dummy data**
  - On server side, reading in only one event
- Future purpose of client
  - Process cells and create memory buffers for cells / imbedding in detector
  - Send direct memory buffer to server to be processed

Performance and resource utilization

- To enhance performance:
  - Load multiple instances onto server
  - Process multiple concurrent requests
- Metrics to evaluate performance:
  - Throughput
  - GPU utilization (often correlated to GPU FLOPs)
  - GPU memory utilization
- Metrics measured with Nvidia's `perf_analyzer` tool

Results

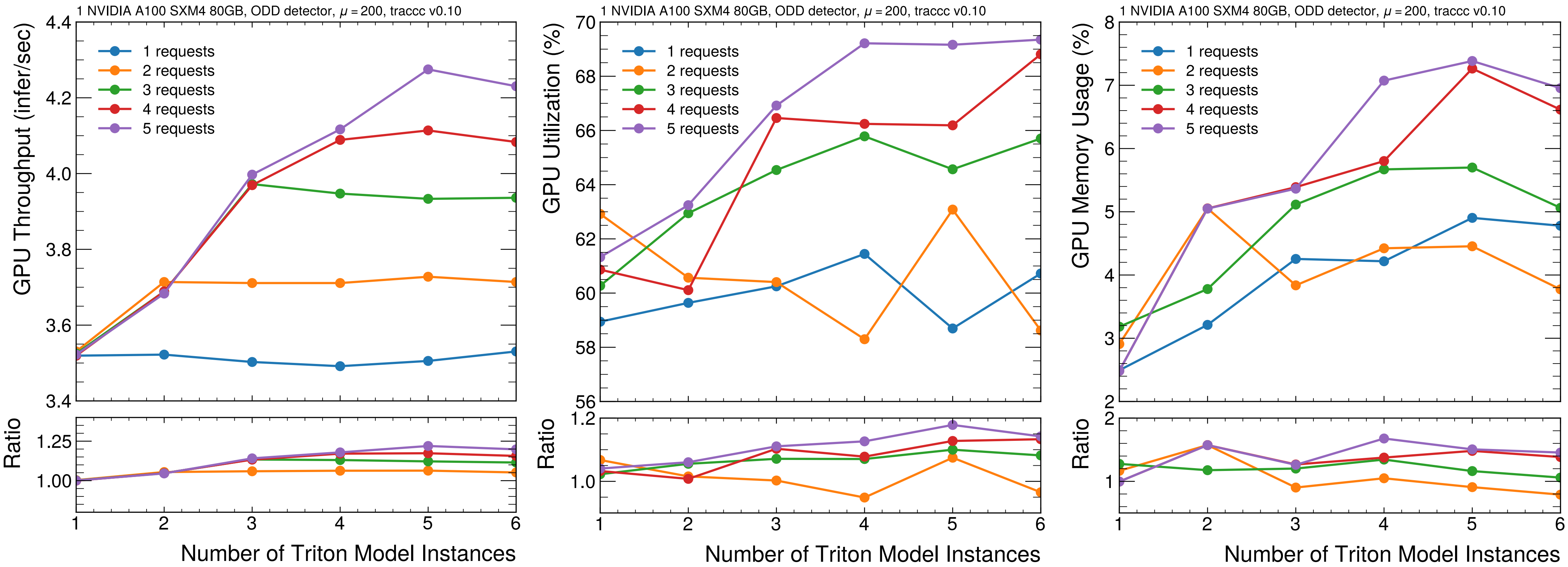


- See good scaling from base performance
- Apparent bottleneck in performance / utilization

Initialize everything on server

- Instead of sending cells to server:
  - Load in cells and embedding in detector during initializing loop
  - Load into device memory
- Should reduce data IO increasing throughput and GPU utilization
  - Expect marginal improvements

Results of initializing everything



- See excellent scaling
- Still some performance we can squeeze out

Summary and next steps

- Presented standalone traccc-as-a-service
- Initial results show good scaling and improved resource utilization
  - Throughput increases from ~2.5 events/sec to ~3.5 events/sec
  - Get this improvement almost for free!
- Next steps:
  - Update to new version of traccc
  - Improve client's abilities to pre-process removing some initialization steps
    - Will replicate real-world model better
  - Match traccc throughput examples (detector caching, multi-threading, etc.)
  - Multi-GPU performance studies
  - Multiple event batching
  - Think about possible integration into athena