Acts-as-a-Service hackathon

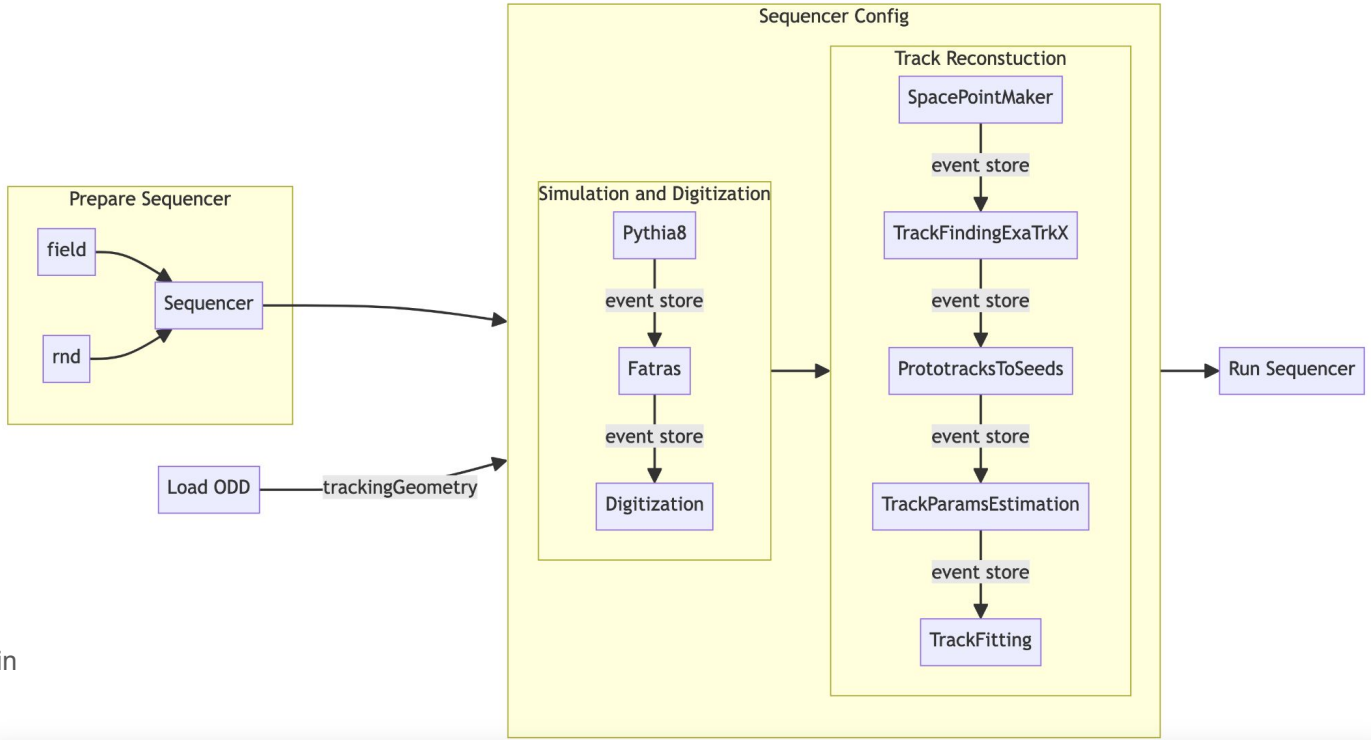
Haoran Zhao, Yuan-Tang Chou, Xiangyang Ju, Elham E Khoda, Andrew Naylor, Yao Yao, Paolo Calafiura, Steven Farrell, Shih-Chieh Hsu, William Patrick McCormack , Philip Coleman Harris , Dylan Sheldon Rankin, Yongbin Feng

ACTS Workshop 2023, Orsay

OAC-2117997

Baseline workflow

exatrkx-acts-demonstrator: [inference.py]

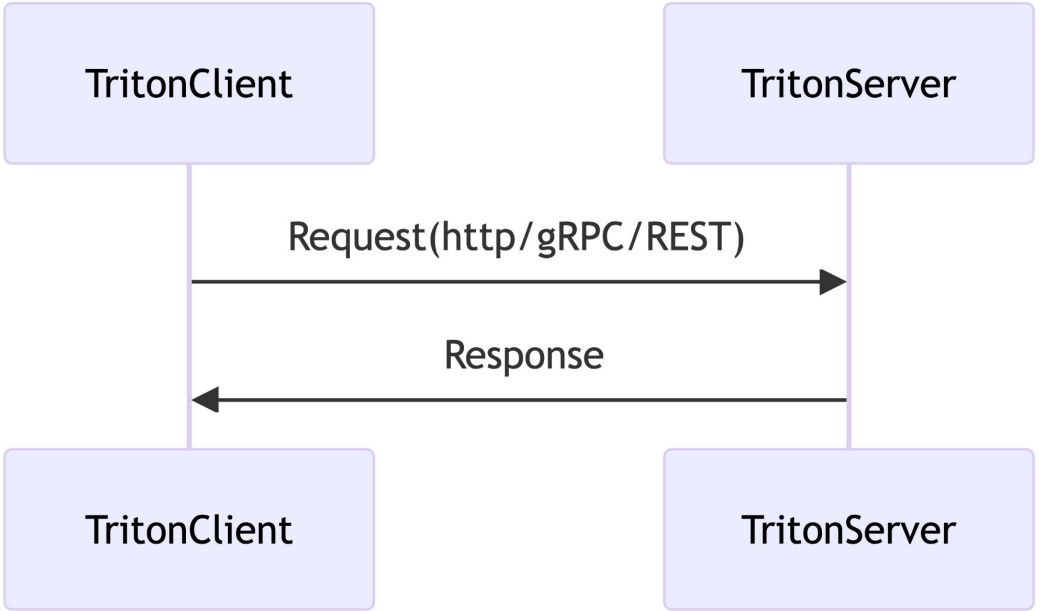


Thanks to Benjamin Huth for the nice demonstrator

Triton Inference Server

Request: inputs for the ML model

Response: inference results



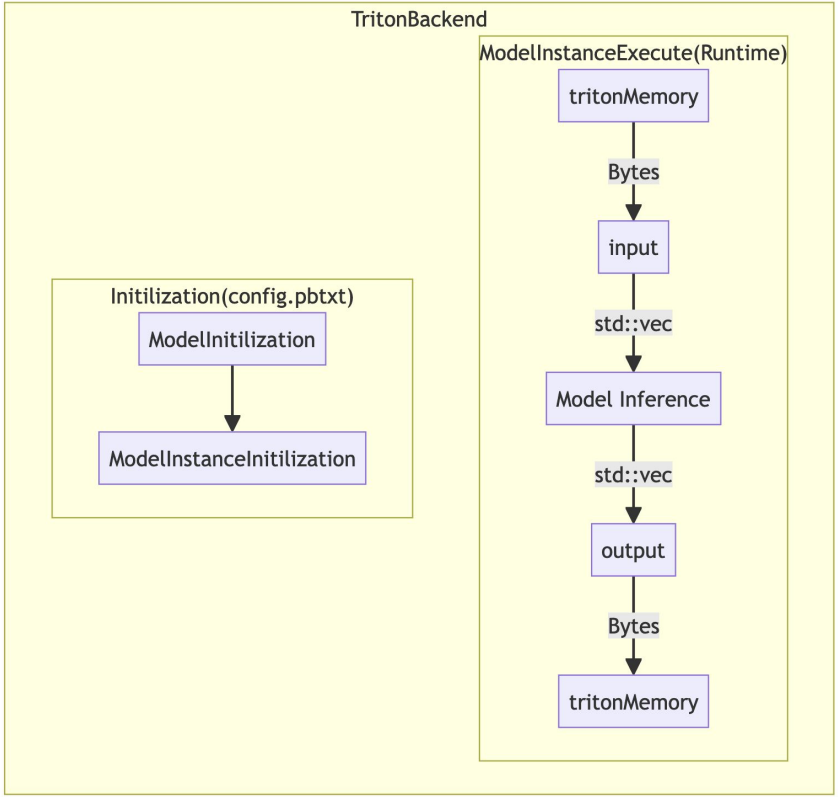
Triton Inference Server Backend

Main components:

- Initialization
- Execution

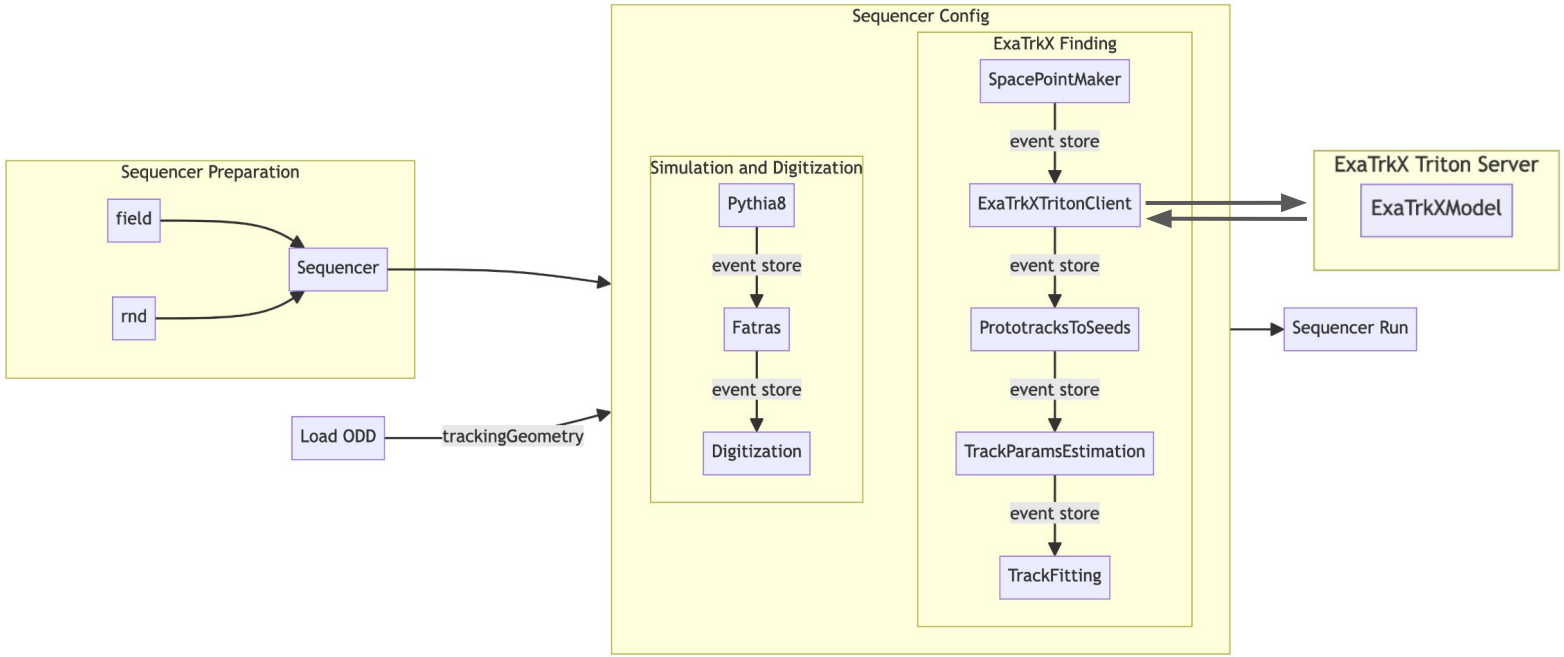
Example:

[exatrkx_gpu backend]



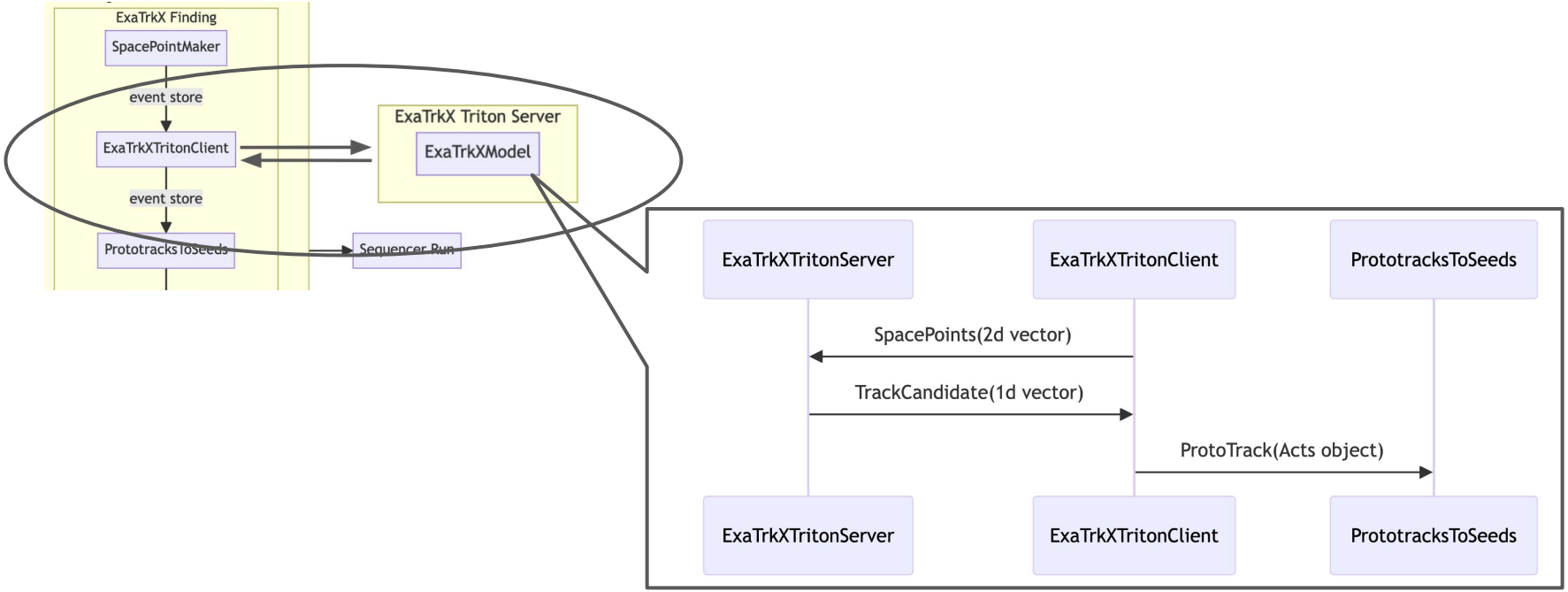
Integration of the ExaTrkX-as-a-Service to ACTS

External ExaTrkX server, see Yuan-Tang's slide



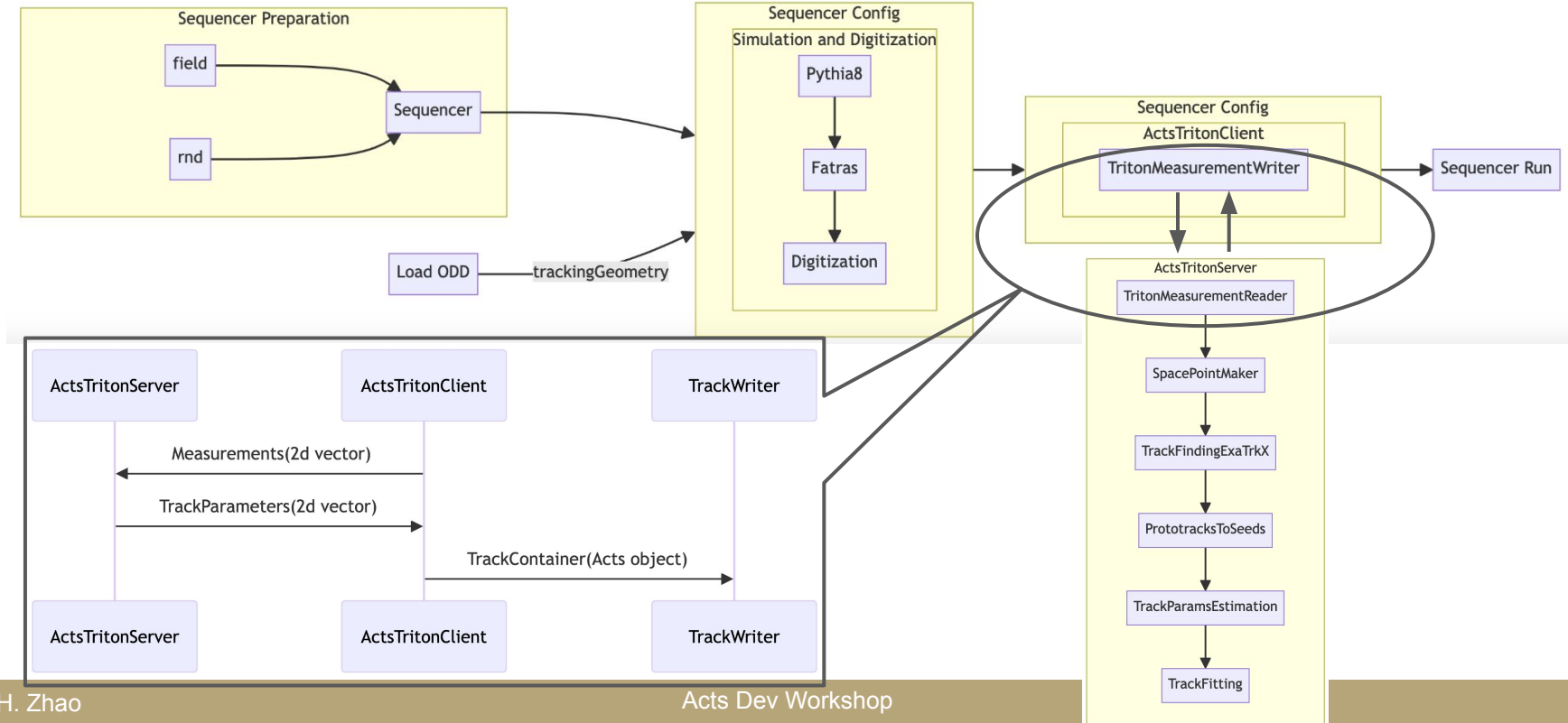
Integration of the ExaTrkX-as-a-Service to ACTS

External ExaTrkX server, see Yuan-Tang's slide



Acts-aaS

Our proposed workflow, measurements in, track out



Discussions

1. I/O between the client and the server
2. Dataflow on the server side
3. Miscellaneous questions

Discussions

1. I/O between the client and the server
   a. What parts does the server do?
      Goal in execution: TrackFinding(GPU) + TrackFitting(GPU) + SpacePointMaker?
      Sourcelinks needed for fitting

      In the initialization: load the ML models, config the B-field, tracking geometry etc..

Discussions

1. I/O between the client and the server
   a. What parts does the server do?
   b. Only the primitive data types are supported;
      Bool, int, float, string

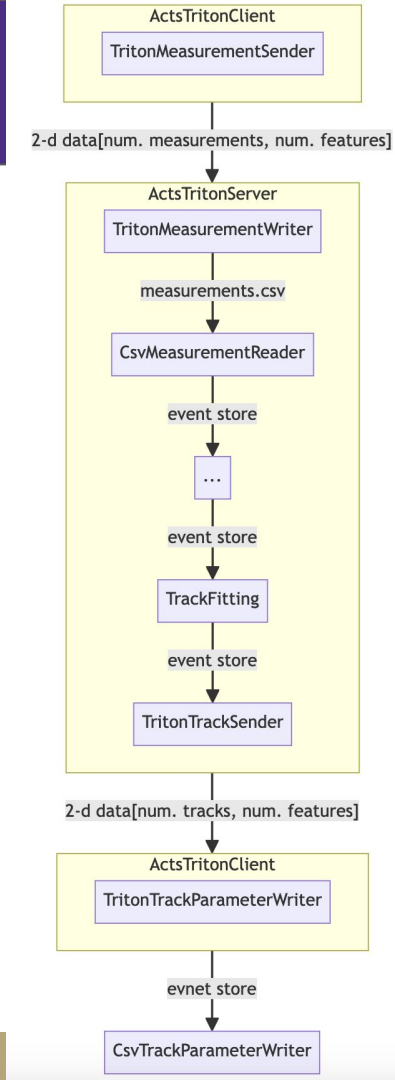
Proposal: send 2-d data with shape [number of measurements, 13]

Wrap to the composite data type on the server

```
measurement_id,geometry_id,local_key,local0,local1,phi,theta,time,var_local0,var_local1,var_phi,var_theta,var_time
0,1152921779484760322,,-0.988179684,27.5145588,0,0,0,9.99999975e-05,9.99999975e-05,0,0,0
1,1152921779484760322,,-1.59969449,25.6581497,0,0,0,9.99999975e-05,9.99999975e-05,0,0,0
2,1152921779484760322,,-7.61218023,25.5525475,0,0,0,9.99999975e-05,9.99999975e-05,0,0,0
3,1152921779484760578,,-8.14472675,28.2101498,0,0,0,9.99999975e-05,9.99999975e-05,0,0,0
```

Discussions

1. I/O between the client and the server
2. Dataflow: Sequencer vs no sequencer option on the server side
   a. Sequencer + python backend
      Port inference.py to the server
      i. Pro: utilises the event store; least development effort
      ii. Con: I/O time with storing csv files



Discussions

1. I/O between the client and the server
2. Dataflow: Sequencer vs no sequencer option on the server side
   a. Sequencer + python backend
   b. No sequencer + custom backend
      i. Pro: reduce the I/O
      ii. Con: code refactoring

```
// the Kalman fitter
ActsExamples::TrackFittingAlgorithm::Config trackFittingConfig;
trackFittingConfig.inputMeasurements = "measurements";
trackFittingConfig.inputSourceLinks = "sourcelinks";
trackFittingConfig.inputProtoTracks = "exatrkx_prototracks";
trackFittingConfig.inputInitialTrackParameters = "trackParameters";
trackFittingConfig.outputTracks = "tracks";

Acts::FreeToBoundCorrection freeToBoundCorrection;
freeToBoundCorrection.apply =
    false;  // Explicitly set to false, though it's the default

std::shared_ptr<ActsExamples::TrackFitterFunction> myKalmanFitter =
    ActsExamples::makeKalmanFitterFunction(trackingGeometry, field, true,
                                           true, 0.0, freeToBoundCorrection);

Acts::CalibrationContext calibContext;
std::shared_ptr<ActsExamples::MeasurementCalibrator> calibrator;
ActsExamples::TrackFittingAlgorithm trackFitting(trackFittingConfig,
                                                 Acts::Logging::INFO);

ActsExamples::ConstTrackContainer consttracksContainer =
    trackFitting.executeTrackFitting(measurements, sourceLinks, protoTracks,
                                     trackParameters, nullptr, geoContext,
                                     magFieldContext, calibContext,
                                     calibrator, myKalmanFitter);
```

Discussions - code refactoring example

TrackFittingAlgorithm.hpp

```
63  /// Framework execute method of the fitting algorithm
64  ///
65  /// @param ctx is the algorithm context that holds event-wise information
66  /// @return a process code to steer the algorithm flow
67  ActsExamples::ProcessCode execute(const AlgorithmContext& ctx) const final;
68
69  /// Refactor the execute method without the need of an AlgorithmContext
70  ///
71  /// @param measurements is the input measurements
72  /// @param sourceLinks is the input source links
73  /// @param protoTracks is the input proto tracks
74  /// @param initialParameters is the input initial track parameters
75  /// @param clusters is the input clusters
76  /// @param geoContext is the geometry context
77  /// @param magFieldContext is the magnetic field context
78  /// @param calibContext is the calibration context
79  /// @param calibrator is the measurement calibrator
80  /// @param fit is the track fitter function
81  /// @return the fitted tracks in a container
82  ConstTrackContainer executeTrackFitting(      You, 2 days ago • adding TrackFi
83      const MeasurementContainer& measurements,
84      const IndexSourceLinkContainer& sourceLinks,
85      const ProtoTrackContainer& protoTracks,
86      const TrackParametersContainer& initialParameters,
87      const ClusterContainer* clusters,
88      Acts::GeometryContext& geoContext,
89      Acts::MagneticFieldContext& magFieldContext,
90      Acts::CalibrationContext& calibContext,
91      std::shared_ptr<MeasurementCalibrator>& calibrator,
92      std::shared_ptr<TrackFitterFunction>& fit) const;
```

TrackFittingAlgorithm.cpp

```
69  ActsExamples::ProcessCode ActsExamples::TrackFittingAlgorithm::execute(
70      const ActsExamples::AlgorithmContext& ctx) const {
71    // Read input data
72    const auto& measurements = m_inputMeasurements(ctx);
73    const auto& sourceLinks = m_inputSourceLinks(ctx);
74    const auto& protoTracks = m_inputProtoTracks(ctx);
75    const auto& initialParameters = m_inputInitialTrackParameters(ctx);
76
77    const ClusterContainer* clusters =
78        m_inputClusters.isInitialized() ? &m_inputClusters(ctx) : nullptr;
79
80    Acts::GeometryContext geoContext = ctx.geoContext;
81    Acts::MagneticFieldContext magFieldContext = ctx.magFieldContext;
82    Acts::CalibrationContext calibContext = ctx.calibContext;
83    std::shared_ptr<MeasurementCalibrator> calibrator = m_cfg.calibrator;
84    std::shared_ptr<TrackFitterFunction> fit = m_cfg.fit;
85
86    auto constTracks = executeTrackFitting(measurements, sourceLinks, protoTracks,
87                                           initialParameters, clusters, geoContext,
88                                           magFieldContext, calibContext, calibrator, fit);
89
90    m_outputTracks(ctx, std::move(constTracks));
91    return ActsExamples::ProcessCode::SUCCESS;
92  }
```

Discussions - Miscellaneous

Surface creation failed w/o geoIDhook(thought optional..)

A global geoContext for odd.py?

odd.py - getOpenDataDetector()

```python
    def geoid_hook(geoid, surface):        Benjamin Huth, 10 months ago • fea
        if geoid.volume() in volumeRadiusCutsMap:
            r = sqrt(surface.center()[0] ** 2 + surface.center()[1] ** 2)

            geoid.setExtra(1)
            for cut in volumeRadiusCutsMap[geoid.volume()]:
                if r > cut:
                    geoid.setExtra(geoid.extra() + 1)

        return geoid

    dd4hepConfig = acts.examples.dd4hep.DD4hepGeometryService.Config(
        xmlFileNames=[str(odd_xml)],
        logLevel=customLogLevel(),
        dd4hepLogLevel=customLogLevel(),
        geometryIdentifierHook=acts.GeometryIdentifierHook(geoid_hook),
    )
    detector = acts.examples.dd4hep.DD4hepDetector()
```

```cpp
class VolumeRadiusGeometryIdentifierHook : public Acts::GeometryIdentifierHook {
 public:
  static const std::unordered_map<unsigned int, std::vector<double>>
      volumeRadiusCutsMap;

  // overwrite decorateIdentifier method
  virtual Acts::GeometryIdentifier decorateIdentifier(
      Acts::GeometryIdentifier identifier,
      const Acts::Surface& surface) const override {
    Acts::GeometryContext geoContext;
    auto volumeIter = volumeRadiusCutsMap.find(identifier.volume());      You, 3 day
    if (volumeIter != volumeRadiusCutsMap.end()) {
      // FIXME: surface needs a GeometryContext, not sure how odd.py does it
      Acts::Vector3 centerPoint = surface.center(geoContext);

      double r = std::sqrt(centerPoint.x() * centerPoint.x() +
                           centerPoint.y() * centerPoint.y());

      unsigned int extraValue = 1;
      for (double cut : volumeIter->second) {
        if (r > cut) {
          extraValue++;
        }
      }
      identifier.setExtra(extraValue);
    }
    return identifier;
  }
};
```

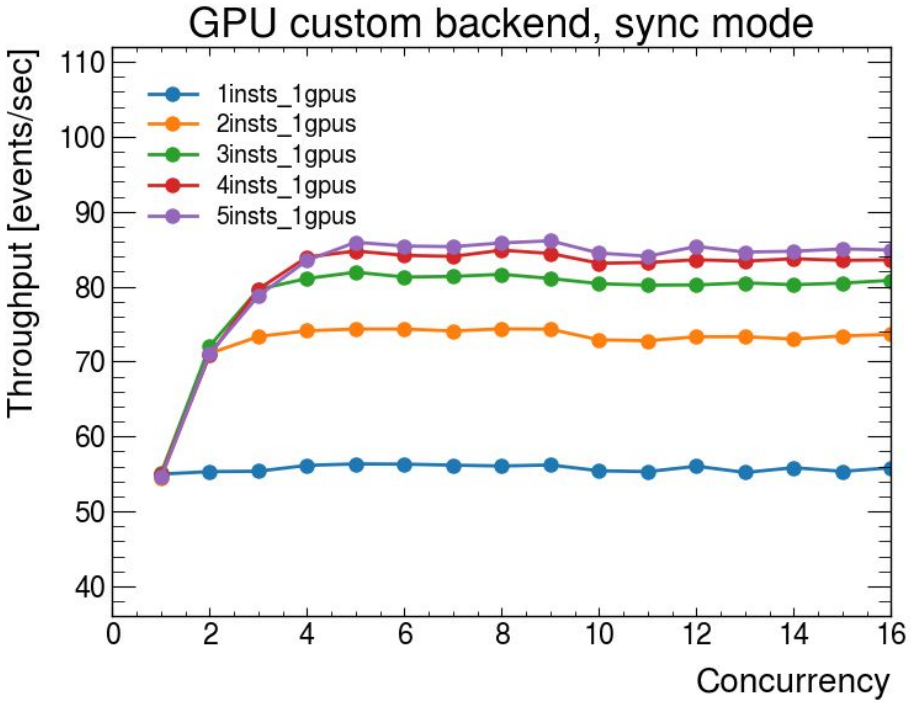
Roadmap

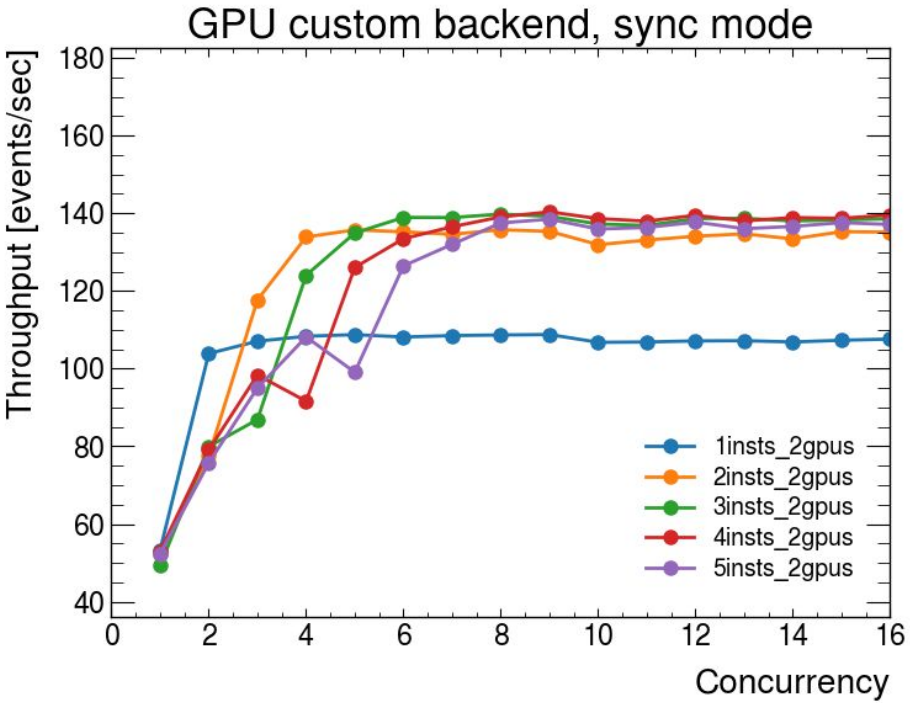
1. Standalone version of cpp
   CsvMeasurementReader -> trackFitting
2. Object Oriented version
3. Triton backend implementation
   a. Develop for I/O between the client and server

Backup

Multiple instances on 1 GPU and 2 GPUs

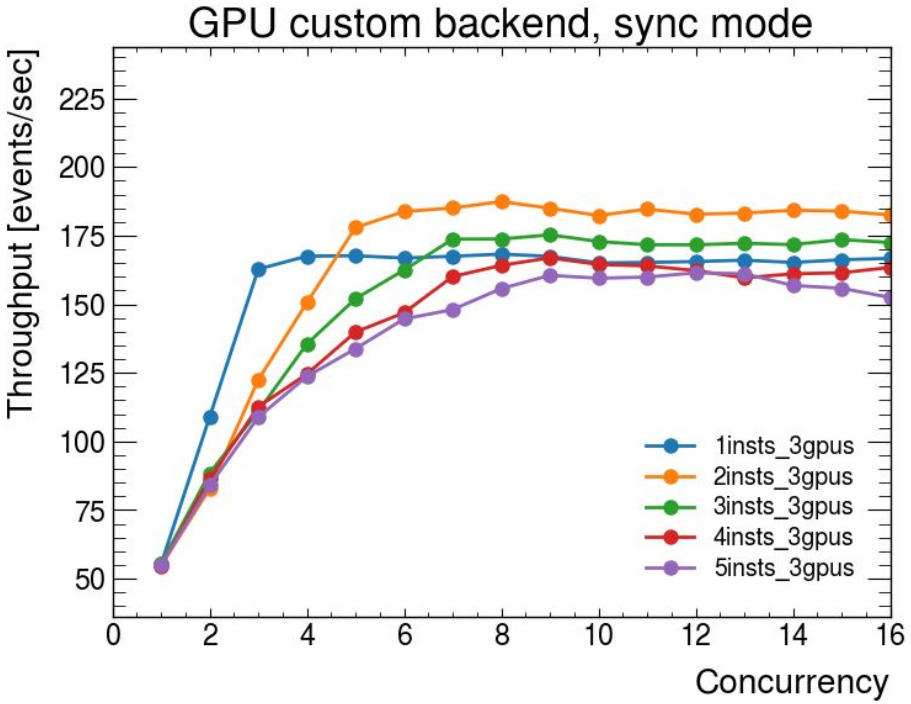
Left plot: 1 GPU max throughput is ~ 80; right plot: scale good with 1 instance

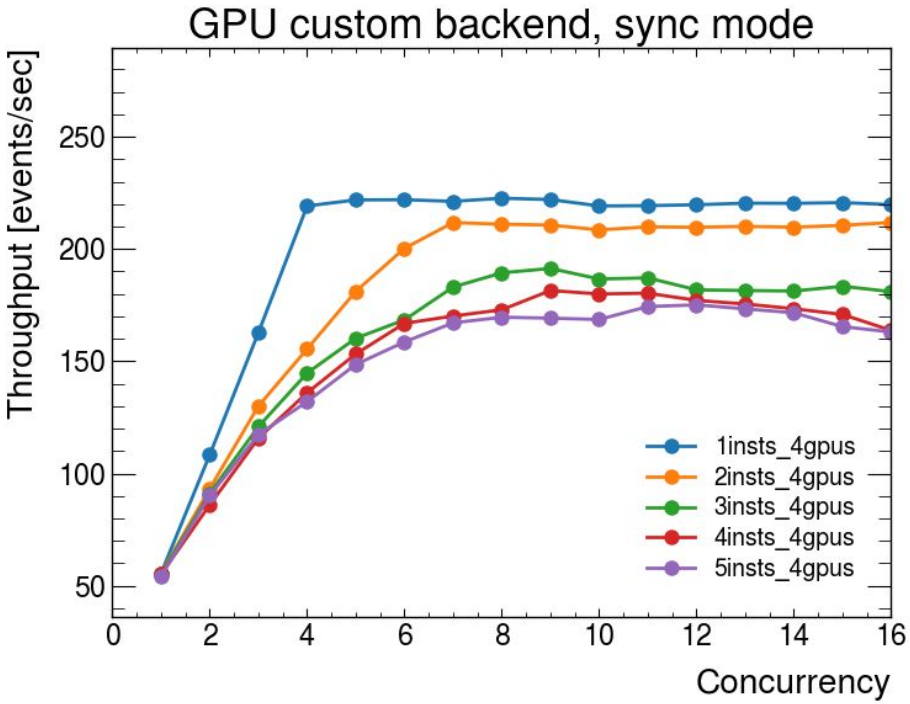




Multiple instances on 3 GPUs and 4 GPUs

Multi instances on multi-GPUs are not scaled properly





Discussions

1. I/O between the client and the server
2. Dataflow: Sequencer vs no sequencer option on the server side
    a. Sequencer + python backend
        i. Pro: least development effort
        ii. Con: I/O time with storing csv files
    b. No sequencer + custom backend
        i. Pro: reduce the I/O
        ii. Con: code refactoring

Acts-aaS

Our proposed workflow, measurements in, track out