This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 871072

Task 3 status: CERN

The 3rd CREMLINplus WP5 general meeting

Plácido Fernández Declara, André Sailer

February 17, 2021

CERN

CREMLIN PLUS
Connecting Russian and European Measures for Large-scale Research Infrastructures

TPC related tools for Aurora integration

- TPC geometry

  https://github.com/iLCSoft/lcgeo/blob/master/detector/tracker/TPC10_geo.cpp

- Geant4 sensitive detector

  https://github.com/iLCSoft/lcgeo/blob/master/plugins/TPCSDAction.cpp

- Digitisation: parametrised resolutions

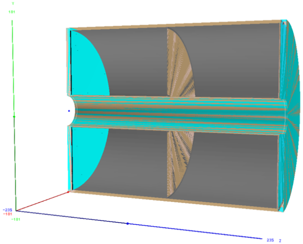
  https://github.com/iLCSoft/MarlinTrkProcessors/blob/master/source/Digitisers/src/DDTPCDigiProcessor.cc

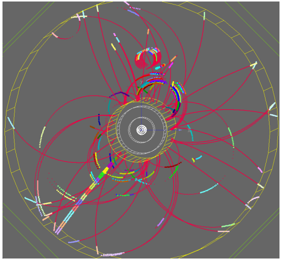
- Pattern recognition and track reconstruction

  https://github.com/iLCSoft/Clupatra

- TPC Reconstruction Steering File

  https://gitlab.cern.ch/sailer/SCT_TPCReco





k4MarlinWrapper

- k4MarlinWrapper brings *Marlin* functionality to *Gaudi* framework, smoothly.
- It creates interfaces *(wraps)* around Marlin Processors, encapsulating them in Gaudi Algorithms.
- Current Marlin source code is kept intact, and it is just called on demand from the Gaudi Framework.

	Marlin	Gaudi
Language	C++	C++
Working unit	Processor	Algorithm
Config. language	XML	Python
Set-up function	init	initialize
Working function	process	execute
Wrap-up function	end	finalize
Transient Data Format	LCIO	*EDM4hep*

k4MarlinWrapper development I

- Bugs were fixed, a manual (`README.md`) was included with instructions to compile, configure, run and test.
- Updated and modernization of the code base.
- Running examples are included as tests.
- A recipe to build it with Spack is also part of the *k4-spack* repo.
- It was included as part of Key4hep, moving there the repo[1].
- CI is now included with GitHub Actions, checking syntax (`clang-format`), build and running tests to keep resilience.

[1] https://github.com/key4hep/

k4MarlinWrapper development II

- Project was integrated into Key4hep, renamed from ”Gaudi-Marlin-Processors” (GMP) to k4MarlinWrapper.
- Gaudi integration was updated to last Gaudi version 35
  - Modern and more standard Cmake
- Spack recipe for it was created. Built and released as part of the Key4hep view.[2]
  - `spack install key4hep-stack`

[2] https://key4hep.github.io/key4hep-doc/spack-build-instructions/README.html

Dependencies

k4MarlinWrapper can be built against the Key4hep CVMFS view. Main dependencies:

- **Gaudi**: to wrap Marlin processors and run the algorithms.
- **Marlin**: to run the underlying processors.
    - It will eventually disappear when only Gaudi Algorithms are used.
- **LCIO**: Event Data Model input/output used by Marlin.
- **EDM4hep**: Event Data Model input/output to be used across the framework.
    - Other data event models could be integrated.
- **k4FWCore**, **k4LCIOReader**, **podio**: leveraging synergies between other Key4hep packages and related.

Other general dependencies:

- **ROOT**, **Boost**

Configuration and running

- Config and running done via Python file as with the Gaudi Framework.
- Processor parameters defined for each instance, and list algorithms configured.
- On algorithm initialization of Marlin Processors, the `MARLIN_DLL` environment variable is used to load the necessary libraries.

```
MyTPCDigiProcessor = MarlinProcessorWrapper("MyTPCDigiProcessor")
MyTPCDigiProcessor.OutputLevel = INFO
MyTPCDigiProcessor.ProcessorType = "DDTPCDigiProcessor"
MyTPCDigiProcessor.Parameters = [
             "DiffusionCoeffRPhi", "0.025", END_TAG,
             "DiffusionCoeffZ", "0.08", END_TAG,
             "DoubleHitResolutionRPhi", "2", END_TAG,
             "DoubleHitResolutionZ", "5", END_TAG,
             "HitSortingBinningRPhi", "2", END_TAG,
             "HitSortingBinningZ", "5", END_TAG,
             "MaxClusterSizeForMerge", "3", END_TAG,
             "N_eff", "22", END_TAG,
             # ...
             ]
algList.append(MyTPCDigiProcessor)
```

XML to Python converter

- A converter from XML steering file to Python options file is available as a Python script.
- It produces the list of Gaudi algorithms, including optional Processors.
  - These are left as commented algorithms that need to be manually uncommented by the user.
  - A comment is also included to indicate its configuration.
  - `# algList.append(MyFastJetProcessor) # Config.OverlayNotFalse`
- It now includes *Constants* parsing from the XML
  - It lists the `CONSTANTS =` to be modified by the user
  - These are replaced in the processors with String substitution: `"%(DD4hepXMLFile_subPath)s" % CONSTANTS`
  - It now supports lists of arguments in the constants as well
- `Marlin -x` can create a steering file containing all the parameters for the known processors. This can be converted to python.

EDM4hep to LCIO conversion

- Conversion between EDM4hep and LCIO needed to run with Marlin Processors
- LCIO to EDM4hep conversion available here: https://github.com/key4hep/k4LCIOReader
- Uses *k4FWCore* to read the input collections indicated in the options file
    - https://github.com/key4hep/k4FWCore

```
from Configurables import k4DataSvc, ToolSvc,
    MarlinProcessorWrapper, EDM4hep2LcioTool
theFile= 'edminput.root'
evtsvc = k4DataSvc('EventDataSvc')
evtsvc.input = theFile

from Configurables import PodioInput
inp = PodioInput('InputReader')
inp.collections = [
    'ParticleIDs',
    'ReconstructedParticles',
    'EFlowTrack'
]
inp.OutputLevel = DEBUG
algList.append(inp)

ToolSvc.LogLevel = DEBUG
```

EDM4hep to LCIO conversion

- Converter implemented in *k4MarlinWrapper* as a Gaudi Tool
- Configured in the options file indicating which processor needs to use the Tool to convert the EDM4hep event to LCIO format
- Events are read through the DataHandle from k4FWCore; these are converted and registered in the Transient Event Store (TES) to make it available to the framework.
- Aurora uses SCT-EDM, which could be integrated.
  - What about the SCT-EDM? Can this be made EDM4hep compatible?

```
MyFastJetProcessor = MarlinProcessorWrapper("MyFastJetProcessor")
...
MyFastJetProcessor.Conversion = [
    # type                                       EDM4hep Collection        LCIO Collection
    "edm4hep::TrackCollection",                  "EFlowTrack",             "Tracks",
    "edm4hep::ReconstructedParticleCollection", "ReconstructedParticles", "PFOCollection"
]
MyFastJetProcessor.addTool(EDM4hep2LcioTool())
...
algList.append(MyFastJetProcessor)
```

Testing

Added testing with `ctest`:

- Simple tests that run Marlin Processors, with and without hits.
- Test that generates an input file with `ddsim`, with actual hits.

```
    ddsim \
        --steeringFile $ILCSOFT/ClicPerformance/HEAD/clicConfig/clic_steer.py \
        --inputFiles $ILCSOFT/ClicPerformance/HEAD/Tests/yyxyev_000.stdhep -N 4 \
        --compactFile $ILCSOFT/lcgeo/HEAD/CLIC/compact/CLIC_o3_v14/CLIC_o3_v14.xml \
        --outputFile $GMP_tests_DIR/inputFiles/testSimulation.slcio
```

- Full CLIC reconstruction is used as test for more complex processors.
- Output checks for regex with `INFO Application Manager Terminated successfully`
- Test to check the XML to Python converter works correctly, with various corner cases.
- Test for the converters between EDM4hep and LCIO, in-memory.

Geometry detector file (I)

TPC geometry, Geant4 sensitive detector, digitisation, pattern recognition & track reconstruction: all pieces build on each other.

- The gas volume is separated into cylinder surfaces, to force *Geant4* to make a step.
    - This in turn produces an energy deposit and hit
- The digitiser gets the hits produced by the simulation, and the geometry information that is contained in the TPC driver
- Reconstruction uses the surfaces that the TPC driver defines

To run simulation and reconstruction for SCTAU using *lcgeo* and *iLCSoft*, add an XML based on the *lcgeo TPC driver* to the SCT detector.

Geometry detector file (II)

Creating the geometry file:

- Based on `sctau_detector_geoinitialize.xml`[3]
- Missing materials were added in `material_mixture.xml`
- Needed constants were added from a mix of sources:
  - `lcgeo/ILD/compact/ILD_common_v01/basic_defs.xml`
  - `lcgeo/ILD/compact/ILD_common_v01/top_defs_common_v01.xml`
  - `lcgeo/ILD/compact/ILD_common_v01/envelope_defs.xml`
  - `lcgeo/ILD/compact/ILD_common_v02/top_defs_ILD_l5_v02.xml`
  - `DD4hep/DDDetectors/compact/detector_types.xml`

[3] https://git.inp.nsk.su/sctau/aurora/-/blob/master/DetectorDescription/DetBase/xml/sctau_detector_geoinitialize.xml

Geometry detector file (III)

- All detectors are added:
    - TPC is replaced by

```
<detectors>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/BeamPipeGeo/beamPipeGeom_def.xml"/>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/FARICHGeo/sctau_FarichPID.xml"/>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/BarrelCrystalCaloGeo/BarrelCrystalCalo.xml"/>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/EndcapCrystalCaloGeo/EndcapCrystalCalo.xml"/>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/CoilGeo/CoilGeom_def.xml"/>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/MuonSystemGeo/MuonSystem_def.xml"/>
  <include ref="../Aurora/0.2.4/InstallArea/x86_64-slc7-gcc9-opt/XML/DriftChamberGeo/DriftChamberGeom_def.xml"/>
  <include ref="my_tpc10_01.xml"/>
</detectors>
```

TPC (I)

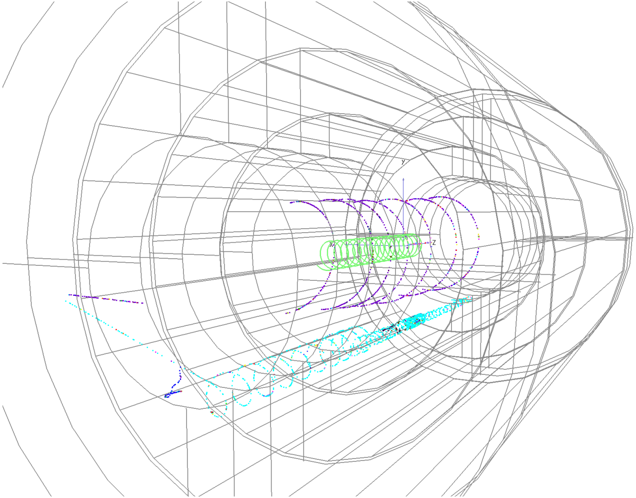
A TPC geometry file is generated [4]:

- It defines the readouts and limits
- The TPC detector is described
  - Dimensions are adapted to match SCTAU's TPC
  - The necessary constants are filled for the *innerWall*, the *outerWall* and the *readout*
- Different things need to be adapted:
  - The composition and thickness of the *innerWall* and *outerWall* is still defined as in ILD

[4] `https://git.inp.nsk.su/plfernan/geom_tpc_aurora`

TPC (II)

- Hits can be visualized by exporting in SLCIO format
-
  ```
  ddsim --compactFile
  my_sctau_det_geo_2.xml -N
  10 -G
  --outputFile=hits.slcio
  --part.userParticleHandler=''
  --gun.isotrop=true
  --gun.energy "100*MeV"
  --steeringFile=steering.py
  ```



Integrating into Aurora

- lcgeo [5] available in Proxima, through Ceph.
- CVMFS could be used to run and compile the different parts.
- A new package for Aurora is in development which now includes the TPC geometry.
- This package will use the TPC Driver from *lcgeo*

[5] `https://github.com/iLCSoft/lcgeo`

SCTAU reconstruction

k4MarlinWrapper successfully computes the full CLIC reconstruction:

- The provided converter can translate to Python Gaudi steering file.
- Algorithms for digitisers, reconstruction, pattern recognition, etc can be included into this sequence.
- The converter adds all algorithms to the list; leaves the configurable ones commented
- Integration with Aurora as an external package:
    - Converters between event data models already in place.
    - Marlin processors can be used as part of the framework through it.
    - This would allow to use TPC digitiser, Clupatra (pattern recognition), track reconstruction & fitting, and the Overlay for background.

Future directions

- We can instantiate the geometry with `ctaurun`
    - Still need to enable the TPCSD
- Then run digitisers, pattern recognition and reconstruction
    - The Marlin file for reconstruction can be converted with the k4MarlinWrapper script
    - Adapters for Event Data Model

Conclusions

- Main achievements
  - Simulation of the lcgeo::TPC in the SCT detector with ddsim
  - Developments for the k4MarlinWrapper to run the processors inside Gaudi, including on the fly conversion of event data
- The objectives for the coming year
  - Fully integrate simulation into the Aurora framework
  - Add the reconstruction for the TPC
- Reinforced collaboration
  - Closer collaboration on core software with BINP inside the Key4hep activities including adption of EDM4hep, Gaudi v35, spack based installations
- Milestones and Deliverables
  - Month 18: Develop software for design of SCT detector