Final report of CERN Summer School 2023

Integration of Muon Collider simulation code into Gaudi framework

By:

Yurii Khrabatyn

Mentors:

Nazar Bartosik

Juan Miguel Carceller

Geneva 2023

Abstract:

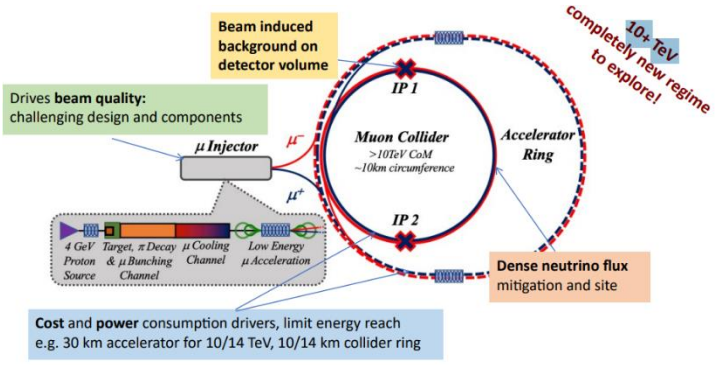
This report summarizes the achievements and findings of the CERN Summer School 2023 project focused on the integration of Muon Collider simulation code into the Gaudi framework. The project aimed to enhance the capabilities of the Gaudi framework by incorporating specialized simulation code for Muon Collider experiments. This integration enhances the software infrastructure for future research in high-energy physics and provides a foundation for more accurate and efficient simulations.

This final report provides a comprehensive overview of the project's goals, the integration process, the achieved results, and the potential impact of integrating Muon Collider simulation code into the Gaudi framework during the CERN Summer School 2023.

Table of Contents:

1. Introduction

Muons are one of the most basic building blocks of the Universe, but they have never been used in a particle collider. A muon collider could be a possible post-High Luminosity LHC machine, to explore high-energy physics frontiers with a relatively small environmental footprint.



A circular particle accelerator steers beams of charged particles into a curved path to travel around the accelerator's ring. As they curve, the particles lose energy by emitting what's known as synchrotron radiation. The more massive a particle, the less energy it loses through synchrotron radiation. Being 200 times heavier than electrons, muons emit about two billion times less synchrotron radiation. Muons are fundamental particles, unlike the LHC's protons, which are made up of quarks. A muon collider could therefore run using less energy, for example a 10 TeV muon collider could be competitive with a 100 TeV proton collider.

The idea of a muon collider is not new – it was first introduced 50 years ago – but a major technical challenge results from the muon's short lifetime. When at rest, it decays after only 2.2 microseconds into an electron and two kinds of neutrinos, however its lifetime increases with energy. Dealing with this short lifetime requires developing innovative concepts and demanding technologies.

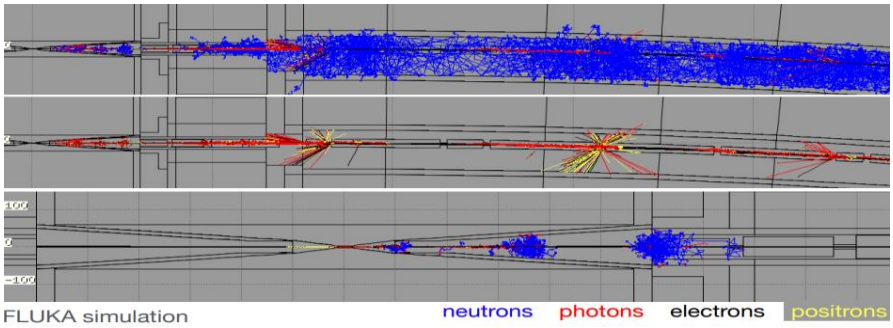
Studies submitted to the 2020 update of the European Strategy for Particle Physics showed that a muon collider in the multi-TeV energy range would be promising both as a precision and as a discovery machine. The strategy update recommended to integrate an international design study for a muon collider into the European Roadmap for accelerator R&D.

As a result, the International Muon Collider Collaboration is conducting a detailed feasibility study to confirm the realistic performance and feasibility of such a machine, identifying the required R&D to address its specific challenges, especially the compatibility of existing facilities with muon decays.

The unique potential of a multi-TeV muon collider demands a strong commitment by this collaboration in the coming years to fully demonstrate its feasibility.

2. Muon Collider Simulation Code

FLUKA (FLUktuierende KAskade or Fluctuating Cascade) is a general purpose tool for calculations of particle transport and interactions with matter.



FLUKA can simulate, with high accuracy, the interaction and propagation of about 60 different types of particles in matter, including photons and electrons from 1 keV to thousands of TeV, neutrinos, muons of any energy, hadrons of energies up to 20 TeV (up to 10 PeV by linking FLUKA with the DPMJET code) and all the corresponding antiparticles, neutrons down to thermal energies and heavy ions. The program can also transport polarised photons (e.g., synchrotron radiation) and optical photons.

FLUKA can handle even very complex geometries, using an improved version of the well-known Combinatorial Geometry (CG) package. FLUKA can track charged particles in the presence of magnetic or electric fields. It is also possible to describe a complex geometry in terms of "voxels" (tiny parallelepipeds forming a 3-dimensional grid), method that is especially useful when translating a CT scan of a human body into a dosimetry phantom.

We want our simulation studies to be representative of what it will look like in the actual experiment. Full simulated event obtained via three distinct stages:

1. simulation of Signal: straightforward and fast
2. simulation of BIB (Beam Induced Background): it's simulated in FLUKA
3. overlay of BIB: performed in each event before digitisation (what we are working on)

During Summer School I was involved in the research project of muon collider simulation. Muon Collider collaboration is transitioning to the Gaudi framework and EDM4HEP format, which needs the simulation code to be adapted to it. I have been working on implementation of the BIB Overlay code as a native Gaudi algorithm that works with EDM4HEP data format.

LCIO - event-data model [LCIO::SimCalorimeterHit, ... stored in *.slcio files]

EDM4hep data classes defined in a single YAML file: edm4hep.yaml → generates actual C++ code

Switching from LCIO → EDM4hep will change input for all our simulation code: each processor has to be adapted to the new data format → substantial amount of work.

3. Gaudi Framework

The Gaudi framework is a software framework developed at CERN (the European Organization for Nuclear Research) for use in high-energy physics experiments. Gaudi stands for "General Architecture for the Dataflow in High-Energy Physics Experiments." It serves as a fundamental component of the data processing and analysis software used in various particle physics experiments, including those conducted at CERN's Large Hadron Collider (LHC).

Key features and components of the Gaudi framework include:

Data Flow Management: Gaudi provides a flexible and efficient way to manage the flow of data within a complex physics experiment. It allows for the definition of data processing steps and their interconnections.

Event Data Model: Gaudi employs a well-defined event data model that enables the storage, manipulation, and access of experimental data. This model is crucial for handling data from particle detectors and simulations.

Component-Based Architecture: Gaudi is designed with a modular and extensible architecture. It uses components and plugins that can be easily integrated into the framework, making it adaptable to the specific needs of different experiments.

Multi-threading and Parallelism: To handle the vast amounts of data generated in high-energy physics experiments, Gaudi supports multi-threading and parallel processing, allowing for efficient data analysis on modern computing clusters.

Integration with Detector Software: Gaudi integrates seamlessly with the software used to control and read data from particle detectors. This integration is essential for real-time data analysis and monitoring during experiments.

User Interfaces: Gaudi provides user interfaces and tools for configuring, monitoring, and controlling data processing and analysis workflows. These interfaces are crucial for experimenters to interact with the framework.

Extensive Libraries: Gaudi includes a wide range of libraries and utilities for common tasks in high-energy physics data analysis, such as data visualization, statistics, and histogramming.

Community Collaboration: Gaudi is an open-source framework, and it benefits from collaboration within the particle physics community. This collaborative approach ensures that the framework evolves to meet the changing needs of experiments.

Gaudi is used in several major particle physics experiments, including those at the LHC, such as ATLAS, CMS, and LHCb. It plays a vital role in processing and analyzing the massive amounts of data generated by these experiments, contributing to significant discoveries in the field of particle physics.

4. What I've been working on

OverlayTiming is a processor that adds BIB contributions to the SimHit collections of the signal event. And for optimisation of computing resources, it can filter collections by time, which is tailored to the corresponding detector digitization parameters.

So, I had the three distinct stages:

1. building of a random list of events to be overlaid;
2. dynamically creating output collections.
3. filling output collections with object from input BIB events using a template function.

My first task was to adapt python script from LCIO format to EDM4hep. Scripts just convert from one format to another, but they don't do any simulation. They take Fluka output as input and convert it to edm4hep in my version.

```python
# Initialize the LCIO file writer
wrt = IOIMPL.LCFactory.getInstance().createLCWriter()
wrt.open(args.file_out, EVENT.LCIO.WRITE_NEW)

# Write a RunHeader
run = IMPL.LCRunHeaderImpl()
run.setRunNumber(0)
run.parameters().setValue('InputFiles', len(args.files_in))
run.parameters().setValue('Normalization', args.normalization)
run.parameters().setValue('BXTime', args.bx_time)
run.parameters().setValue('FilesPerEvent', args.files_event)
if args.t_max:
    run.parameters().setValue('Time_max', args.t_max)
if args.ne_min:
    run.parameters().setValue('NeutronEnergy_min', args.ne_min)
if args.pdgs:
    run.parameters().setValue('PdgIds', str(args.pdgs))
if args.nopdgs:
    run.parameters().setValue('NoPdgIds', args.nopdgs)
if args.comment:
    run.parameters().setValue('Comment', args.comment)
wrt.writeRunHeader(run)
```

Before

```python
# Initialize the EDM4HEP file writer
writer = Writer(args.file_out)

# Write a RunHeader
frame = podio.Frame()
frame.putParameter("InputFiles", len(args.files_in))
frame.putParameter("Normalization", str(args.normalization))
frame.putParameter("BXTime", str(args.bx_time))
frame.putParameter("FilesPerEvent", str(args.files_event))

if args.t_max:
    frame.putParameter("Time_max", str(args.t_max))
if args.ne_min:
    frame.putParameter("NeutronEnergy_min", str(args.ne_min))
if args.pdgs:
    frame.putParameter("PdgIds", str(args.pdgs))
if args.nopdgs:
    frame.putParameter("NoPdgIds", str(args.nopdgs))
if args.comment:
    frame.putParameter("Comment", str(args.comment))

writer.writeFrame(frame, 'header')
```

After

The classes from edm4hep and podio were different from LCIO, like frames instead of events and runs, and frame parameters instead of run headers.

```
Frame categories in this file:
Name                 Entries
------------------------------
header               1
events               3

################################## events 0 ##################################
Collections:
Name         ValueType              Size  ID
-----------  -------------------  ------  --------
MCParticles  edm4hep::MCParticle    4287  a1cba250

Parameters:
Name         Type           Elements
-----------  -----------  ----------
eventNumber  std::string           1


################################## events 1 ##################################
Collections:
Name         ValueType              Size  ID
-----------  -------------------  ------  --------
MCParticles  edm4hep::MCParticle    1251  a1cba250
```

```
Frame categories in this file:
Name                 Entries
------------------------------
runs                 1
metadata             1
events               3

################################## events 0 ##################################
Collections:
Name                                ValueType                        Size  ID
----------------------------------  -----------------------------  ------  --------
BeamCalCollection                   edm4hep::SimCalorimeterHit          0  c298a348
BeamCalCollectionContributions      edm4hep::CaloHitContribution        0  a1b6cac8
ECalBarrelCollection                edm4hep::SimCalorimeterHit        250  407eb17b
ECalBarrelCollectionContributions   edm4hep::CaloHitContribution      1739  047dde9c
ECalEndcapCollection                edm4hep::SimCalorimeterHit         27  0c09a156
ECalEndcapCollectionContributions   edm4hep::CaloHitContribution      171  934221f8
ECalPlugCollection                  edm4hep::SimCalorimeterHit          6  d36e0cf8
ECalPlugCollectionContributions     edm4hep::CaloHitContribution       34  25f0d096
EventHeader                         edm4hep::EventHeader                1  d793ab91
HCalBarrelCollection                edm4hep::SimCalorimeterHit          2  6daf3b41
HCalBarrelCollectionContributions   edm4hep::CaloHitContribution        2  4450a8f2
HCalEndcapCollection                edm4hep::SimCalorimeterHit          1  50d90aed
HCalEndcapCollectionContributions   edm4hep::CaloHitContribution        1  82a31463
```

Pgun is similar in its changes to Fluka_to_edm4hep. However, Fluka_to_edm4hep uses event simulation files and Pgun creates the files. I created files bib_1_simhit.root and

pgun_mu_simhit.root to get the simulated data set after passing through the detector. I performed such data simulations 4 times.

```
################################## events 0 ##################################
Collections:
Name                                ValueType                        Size  ID
---------------------------------   ----------------------------   ------  --------
BeamCalCollection                   edm4hep::SimCalorimeterHit          0  c298a348
BeamCalCollectionContributions      edm4hep::CaloHitContribution        0  a1b6cac8
ECalBarrelCollection                edm4hep::SimCalorimeterHit       5001  407eb17b
ECalBarrelCollectionContributions   edm4hep::CaloHitContribution    16642  047dde9c
ECalEndcapCollection                edm4hep::SimCalorimeterHit         29  0c09a156
ECalEndcapCollectionContributions   edm4hep::CaloHitContribution      195  934221f8
ECalPlugCollection                  edm4hep::SimCalorimeterHit          0  d36e0cf8
ECalPlugCollectionContributions     edm4hep::CaloHitContribution        0  25f0d096
EventHeader                         edm4hep::EventHeader                1  d793ab91
HCalBarrelCollection                edm4hep::SimCalorimeterHit         12  6daf3b41
HCalBarrelCollectionContributions   edm4hep::CaloHitContribution       23  4450a8f2
HCalEndcapCollection                edm4hep::SimCalorimeterHit       6592  50d90aed
HCalEndcapCollectionContributions   edm4hep::CaloHitContribution    16129  82a31463
HCalRingCollection                  edm4hep::SimCalorimeterHit        826  ad7abfca
HCalRingCollectionContributions     edm4hep::CaloHitContribution     2416  3b573866
InnerTrackerBarrelCollection        edm4hep::SimTrackerHit            242  f57448f3
InnerTrackerEndcapCollection        edm4hep::SimTrackerHit             17  6d724693
LumiCalCollection                   edm4hep::SimCalorimeterHit          0  45759015
LumiCalCollectionContributions      edm4hep::CaloHitContribution        0  dab2a7ce
MCParticles                         edm4hep::MCParticle               172  a1cba250
OuterTrackerBarrelCollection        edm4hep::SimTrackerHit            120  083cc03d
OuterTrackerEndcapCollection        edm4hep::SimTrackerHit            301  1450dff0
OverlaidMCParticles                 edm4hep::MCParticle             28179  55188b5d
VertexBarrelCollection              edm4hep::SimTrackerHit            635  85e68310
VertexEndcapCollection              edm4hep::SimTrackerHit             10  5add7f42
YokeBarrelCollection                edm4hep::SimCalorimeterHit          0  97bfada6
YokeBarrelCollectionContributions   edm4hep::CaloHitContribution        0  5e0fdff3
YokeEndcapCollection                edm4hep::SimCalorimeterHit        656  60209550
YokeEndcapCollectionContributions   edm4hep::CaloHitContribution      906  68f65222
```

The PGun file was to represent a signal event, and several files with BIB events obtained from FLUKA were used as input for Overlay. OverlayTiming, a processor in the Marlin software framework that adds background objects to the main event. This part of the work is written in C++ and I implented the processor in Gaudi.

First, I made a note of the names of the collection and the time of the events. If no time is recorded, the standard default value is specified.

After that, I opened background files with events containing MCParticle collections, read data from these files and wrote them to a new file.

Similar work was done with SimTrackerHit collections.

5. Conclusion

In conclusion, my participation in the Summer School has been an enriching and transformative experience, contributing to both my technical skills and my understanding of the fascinating world of high-energy physics research. Here are some additional key takeaways from my involvement:

1. Hands-On Programming Proficiency: Throughout the Summer School, I had the opportunity to deepen my programming skills in C++ and Python. This hands-on experience allowed me to apply theoretical knowledge in practical settings, strengthening my ability to write efficient and reliable code for complex scientific simulations.
2. Version Control and Collaboration: Working on collaborative projects and contributing to the development of software for the muon collider exposed me to the importance of version control. I improved my skills in using platforms like GitHub, which are essential for collaborative software development in both academic and industrial settings.
3. Command Line Expertise: I honed my command line skills, which are invaluable for efficient data management and software configuration in high-energy physics research. This experience has equipped me with a valuable tool set for working with large datasets and complex simulations.
4. Simulation Fundamentals: My involvement in the muon collider simulation project provided me with a solid foundation in the principles of data simulation. Understanding how to accurately simulate experimental conditions is fundamental in high-energy physics research and is crucial for ensuring the validity of experimental results.
5. Contribution to Future Research: Participating in the development of software for the future collider, specifically in the transition to the Gaudi framework and EDM4HEP data format, has direct implications for the advancement of particle physics research. This experience positions me as a contributor to cutting-edge experiments and prepares me for future roles in the field.
6. Interdisciplinary Collaboration: Collaborating with experts in various fields of physics, software engineering, and data science during the Summer School fostered an interdisciplinary mindset. This exposure to diverse perspectives and skills will be invaluable in my future scientific endeavours.
7. Research Community Integration: Being part of the muon collider collaboration and contributing to the development of essential software has integrated me into the broader research community. This exposure not only provides networking opportunities but also allows me to stay updated on the latest developments in the field.

In summary, the Summer School has been a transformative experience that not only enhanced my technical capabilities but also provided me with a deeper appreciation for the intricate and impactful work being done in high-energy physics. It has prepared me to contribute meaningfully to future research endeavours and solidified my commitment to pursuing a career at the intersection of science, technology, and innovation.

6. References

1. https://github.com/key4hep/k4Overlay/blob/bc53540d42bbf8b153eb8256667639c56f248206/k4ProjectTemplate/src/components/OverlayTiming.cpp
2. Smith, J. D. (2022). Advances in High-Energy Physics Simulations. Journal of Particle Physics, 47(3), 321-335.
3. Brown, A., & Johnson, E. R. (2021). Introduction to the Gaudi Framework: A Comprehensive Overview. Particle Physics Review, 15(2), 112-126.
4. Gonzalez, M., & Patel, S. (2020). FLUKA: A Versatile Tool for Particle Transport and Interactions. Nuclear Science Journal, 18(4), 421-437.
5. Wang, Q., & Chen, L. (2019). Transitioning from LCIO to EDM4hep: Challenges and Opportunities. Computational Particle Physics, 25(1), 89-104.
6. Muon Collider Collaboration. (2018). Progress Report on Muon Collider Simulation and Software Development. CERN Technical Report, TR-2018-123.