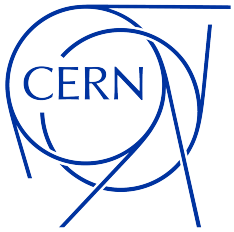



**CERN Summer Student Report**

# Porting and validation of reconstruction algorithms to the Key4hep framework

**Katerina Kostova**

Supervisors:
**Juan Miguel Carceller and Swathi Sasikumar**

27 September 2024

**Abstract**

Key4hep is a turnkey software offering a common platform for future collider experiments. The main idea is to share the existing tools and technologies to reduce duplication and waste of resources and allow everyone to benefit from the newest improvements. In this project, two reconstruction algorithms developed by the International Linear Collider (ILC) community were ported to the Key4hep framework. Validation of the ported algorithms was done as this is an important step in the integration of components into a new framework. Following the successful porting and validation of the algorithms, they were integrated into the Key4hep framework and made available for use in the upcoming studies for future experiments.

# Contents



# 1 Introduction

Key4hep [1] is a turnkey software for future colliders that is currently in the process of development. It provides a full experiment life-cycle, meaning that everything from generation to physical analysis will be included in it. The main components of the Key4hep framework are displayed on Figure 1

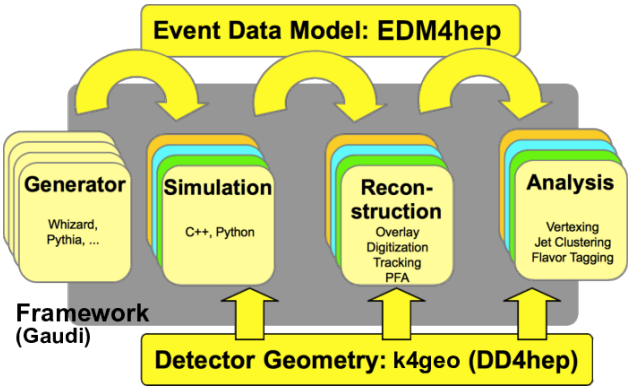



Figure 1: Key4hep structure and main components [2]

With the uncertainty of which $e^+e^-$ collider will be built next, the probability of duplication of efforts and waste of resources is very high on the individual level. For this reason, communities from various future experiments like CLIC, ILC, FCC, Muon Collider, etc. are coming together for the development of Key4hep as a common software framework. The motivation is to share different components in order to reduce the cost of maintenance and to allow everyone to benefit from them.

A notable example is the International Linear Collider (ILC) community, which has been developing tools and algorithms within the iLCSoft framework [3] for over two decades. To integrate these components into Key4hep, they must be adapted to utilize Key4hep's processing framework, Gaudi, as the original tools were developed within the Marlin framework.

Currently, these algorithms can be run in Key4hep with the help of the so-called MarlinProcessorWrapper [4]. It allows the running of Marlin processors in Gaudi by converting all files from LCIO (the International Linear Collider input-output data format) to EDM4hep (the event data model of Key4hep). Nevertheless, the motivation for porting these processors to the native Key4hep framework is to reduce runtime, save computational resources, and, most importantly, make them a generic tool for all future experiments.

To ensure the integrity and reliability of the algorithms, it is essential to verify their performance and confirm that no unintended changes were introduced during the porting process.

## 2 Workflow of the project

For achieving the goals of this project there are several important steps shown schematically in Figure 2 and explained below.

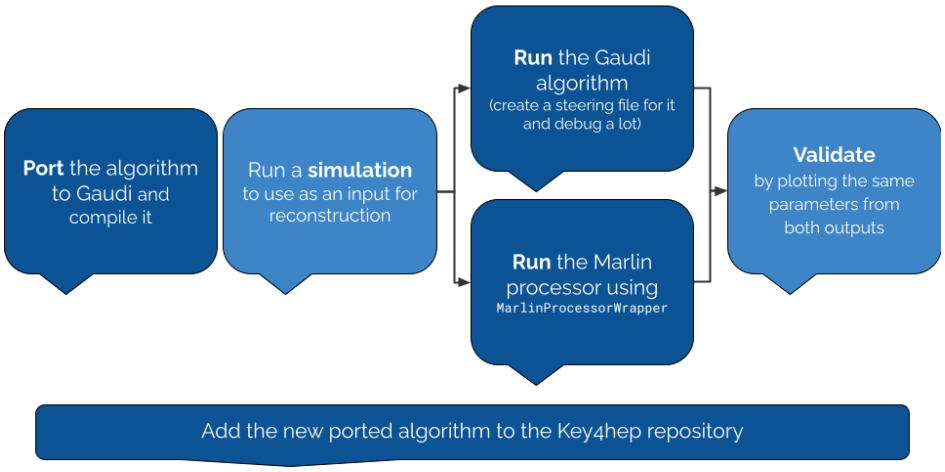



Figure 2: Workflow of the project

1. **Port** the reconstruction algorithm to Gaudi and compile it. The porting process involves replacing the Marlin framework with Gaudi where needed. To compile the algorithm it's required to source the Key4hep stack, build and install the algorithm:

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
# or source /cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh
k4_local_repo
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=../install -G Ninja
ninja install
```

2. Run a **simulation**. Events of particles from a particle gun are simulated for a chosen detector. An example of the simulation code is shown below, where the detector can be set, as well as the particle type, energy and distribution. The simulated data is then used as an input for the reconstruction algorithm.

```
ddsim --compactFile
            $K4GEO/FCCee/CLD/compact/CLD_o2_v06/CLD_o2_v06.xml \
  --enableGun \
  --gun.distribution uniform \
  --gun.energy "10*GeV" \
  --gun.particle gamma \
  --gun.multiplicity 1 \
  --numberOfEvents 1000 \
  --gun.phiMin 0.00000 \
  --gun.phiMax  6.28319 \
  --gun.thetaMin  0.00000 \
  --gun.thetaMax 3.14159 \
  --outputFile sim_partgun_1000.root
```

3. **Run** the algorithm ported to Gaudi. Create a steering `.py` file and remove all unnoticed errors. To run the algorithm in Ke4hep the following command is executed:

```
k4run repository_name/steering_file_name.py
```

4. **Run** the Marlin processor using the `MarlinProcessorWrapper` to serve as a reference for validation. To achieve this the same command as above is executed, but with the steering file of the Marlin processor.

5. **Validate** the ported algorithms by plotting and comparing the same parameters from the ported Gaudi algorithm and the original Marlin processors.

6. **Add** the ported algorithm to the Key4hep repository as a last step after a successful porting and validation.

# 3 The reconstruction algorithms for porting

For this project, an iLCSoft package called DDMarlinPandora was chosen to be ported from the Marlin framework to Gaudi. This work led to the development of a new package within the Key4hep software, named k4GaudiPandora. This package includes the digitisation algorithms DDSimpleMuonDigi and DDCaloDigi, which were the focus of this work. Both of these algorithms were already partly ported by S.Sasikumar.

Digitisation algorithms are part of the reconstruction process that is used to prepare the data for physical analysis. These algorithms take the "raw data" generated by a simulation

and transform it into what a real digital signal from the detector would look like. The core part of the digitisation algorithm is to correct various detector effects. This is done by applying calibration, setting thresholds, making time corrections, etc.

## 3.1 DDSimpleMuonDigi

The first algorithm for porting and validation is DDSimpleMuonDigi. This is a simple digitisation algorithm mainly for muons. In its core part it is doing the following:

- Multiply the energy of every simulated calorimeter hit by a calibration coefficient and get the (reconstructed) calorimeter hit energy;
- Check if the calorimeter hit energy > max hit energy, then write the calorimeter hit energy equal to the max hit energy;
- Check if the calorimeter hit energy > energy threshold, then save all the information about this hit.

### 3.1.1 Porting and running DDSimpleMuonDigi

After successfully compiling the ported algorithm without errors, a simulation of 1000 events of a single muon particle gun with 10 GeV energy in the CLD detector was run to serve as input data for this algorithm.

For running DDSimpleMuonDigi the collections of hits from muon found in the Yoke Barrel and the Yoke Endcap parts of the detector were used. These parts form the muon system of the CLD detector and are shown as Yoke in Figure 3.

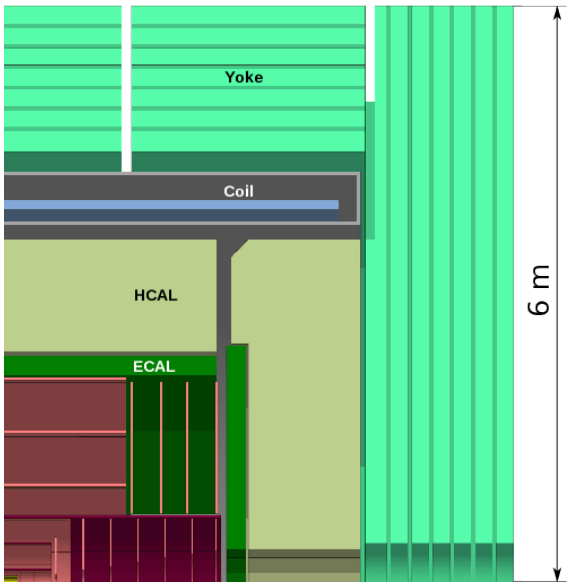



Figure 3: The CLD detector concept [5]

### 3.1.2 Validation of DDSimpleMuonDigi

The most important parameter for a digitisation algorithm is the energy of the digitised hits. That's why it's essential to compare the distribution of the calorimeter hit energy as an output parameter from the Marlin processor (iLCSoft) and the Gaudi algorithm (Key4hep). As seen in Figure 4, both histograms perfectly overlap on top of each other, making it evident that the porting was successful.

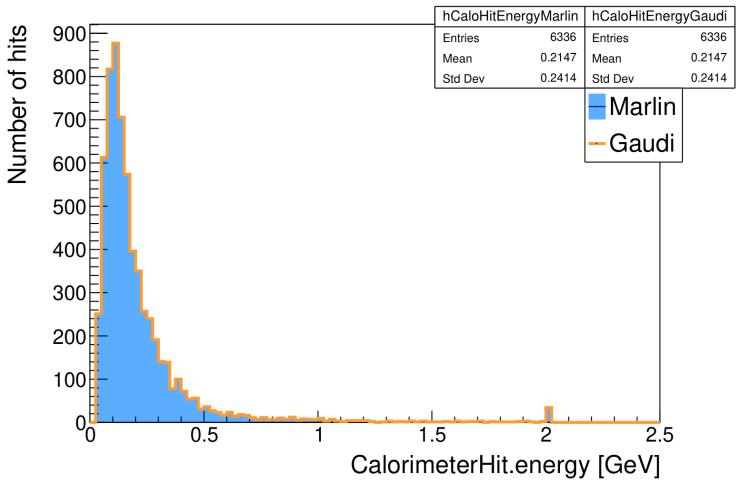



Figure 4: Distribution of the energy of calorimeter hits from DDSimpleMuonDigi.

Another key parameter for digitisation algorithms is the sum of the energy of the calorimeter hits of one particle per event. For the investigation of this parameter, the simulation was done with one single particle per event. In this way, the energy of all the hits in one event can be summed up as they belong to only one particle. This parameter can be interpreted as the deposited energy of the particle in the Yoke system. In principle, the sum of the energy of the calorimeter hits can be used for calibration of the detector. In the case of muons, it's not so relevant, because muons being minimally ionising particles do not completely deposit their energy in the muon system and very often they escape the detector. Therefore more methods are needed to obtain the reconstructed particle energy and to do the calibration for this detector part. The distribution of the sum of the energy of the calorimeter hits per event is shown in Figure 5 from both the Marlin processor (iLCSoft) and Gaudi algorithm (Key4hep). As seen the two histograms are the same as expected when the porting was done correctly.

In summary, the objectives of this project were achieved with the finalisation of porting the DDSimpleMuonDigi algorithm to the Gaudi framework. The algorithm was successfully compiled and run. Validation of the ported algorithm was performed by comparing key output parameters between the Marlin processor (iLCSoft) and the Gaudi algorithm (Key4hep). Following successful validation, a pull request (PR) was submitted to the k4GaudiPandora repository, and the ported algorithm has since been merged and integrated into the Key4hep software stack, making it available for use by the Key4hep community.

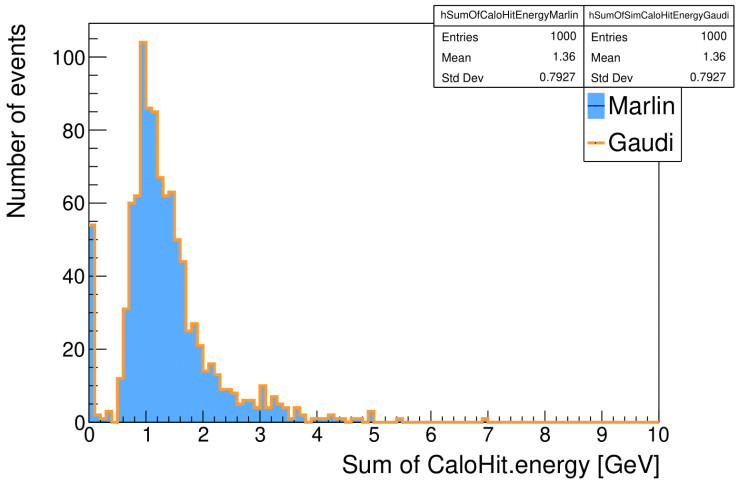



Figure 5: Distribution of the sum of the energy of calorimeter hits from DDSimpleMuonDigi.

## 3.2 DDCaloDigi

The second digitisation algorithm that was the focus of this work is DDCaloDigi. It is a more complex algorithm aiming to digitise hits found in the electromagnetic (ECAL) and hadronic (HCAL) calorimeters of the detector system. Similar to the DDSimpleMuonDigi, DDCaloDigi sets up an energy threshold for the calorimeter hits and applies a calibration constant respective to the given calorimeter part. Additionally, timing cuts, as well as some other digitisation effects for the scintillator and silicon parts are applied.

### 3.2.1 Porting and running DDCaloDigi

Similar to DDSimpleMuonDigi, the goal here is to finalise the nearly ported algorithm DDCaloDigi, ensure that every part of it is included and it compiles without any errors, along with the validation process.

Once the framework has been fully transitioned to Gaudi and no errors are encountered during the compilation process, the next step is to run the algorithm. This requires a simulation file to serve as the input. The simulation for DDCaloDigi is similar to the one for DDSimpleMuonDigi, with the difference that the particle gun here is of photons.

### 3.2.2 Validation of DDCaloDigi

DDCaloDigi is meant to be a calorimeter digitisation algorithm for both the ECAL and the HCAL. These parts of the detector can be seen in Figure 3. All ECAL and HCAL collections of hits from the simulation are included when running the algorithm to perform a full validation.

The first relevant parameter in the validation is again the energy of the digitised calorimeter hits. The distributions obtained from the Marlin processor (iLCSoft) and the Gaudi algorithm (Key4hep) are shown in Figure 6. It is visible that they overlap perfectly, which means that the porting was successful.

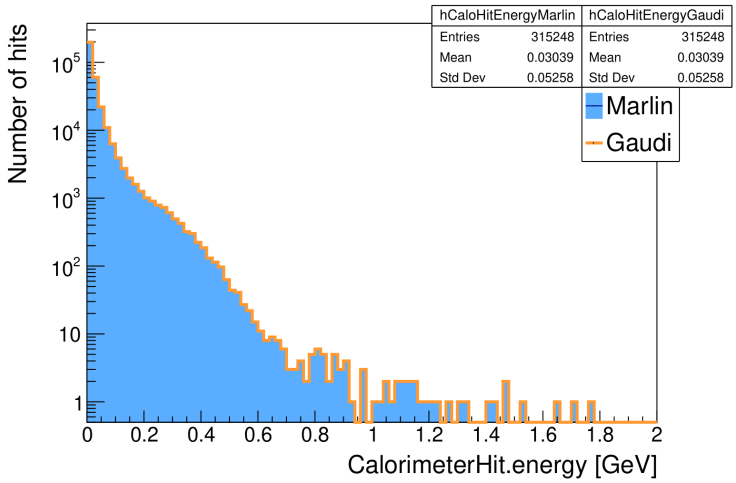
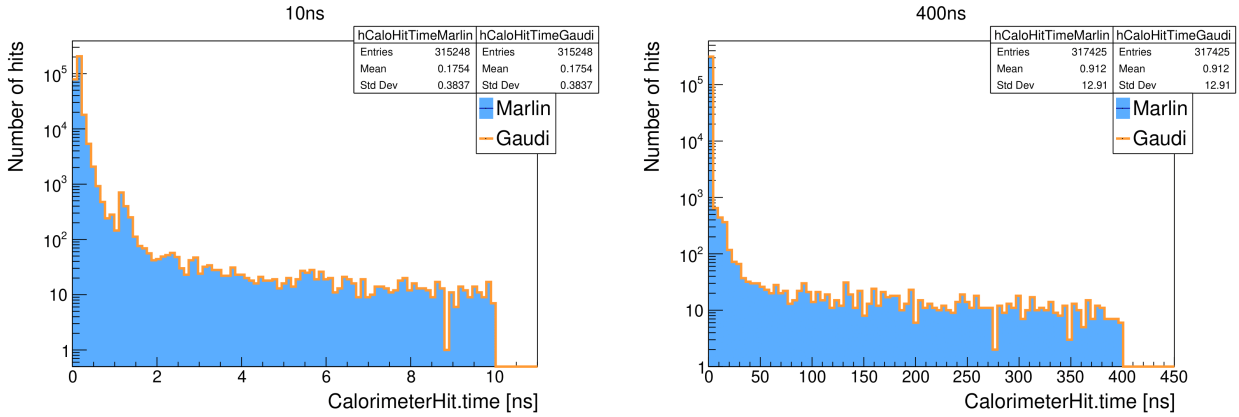



Figure 6: Distribution of the energy of calorimeter hits from DDCaloDigi.

Another important parameter for this algorithm is the calorimeter hit time. A significant part of the code of the DDCaloDigi algorithm is for applying a timing cut to the calorimeter hits, so it's essential to ensure that everything in this part of the digitisation process works correctly after the porting. The plots in Figure 7 show the distribution of the calorimeter hit time for two different values of the timing cut: 10 ns and 400 ns. As seen the distributions from the Marlin processor (iLCSoft) and the Gaudi algorithm (Key4hep) are the same, following that everything works correctly after the porting.




(a) for timing cut = 10 ns

(b) for timing cut = 400 ns

Figure 7: Distribution of the time of calorimeter hits from DDCaloDigi.

Lastly, the sum of the energy of the calorimeter hits per event is validated. This is an essential parameter for the digitisation of calorimeters since it can be used for calibration. If the particle is fully absorbed in the calorimeter, then the sum of the energy of the calorimeter hits of the particle per event should have a Gaussian distribution around the simulated particle energy (which in this case is 10 GeV). From the plot in Figure 8 can be seen that this is mostly the case for this algorithm. This means that the digitisation is working as

expected. The validation is also shown on the same plot. The output from both the Marlin processor (iLCSoft) and the Gaudi algorithm (Key4hep) is the same as it should be when porting is successful.

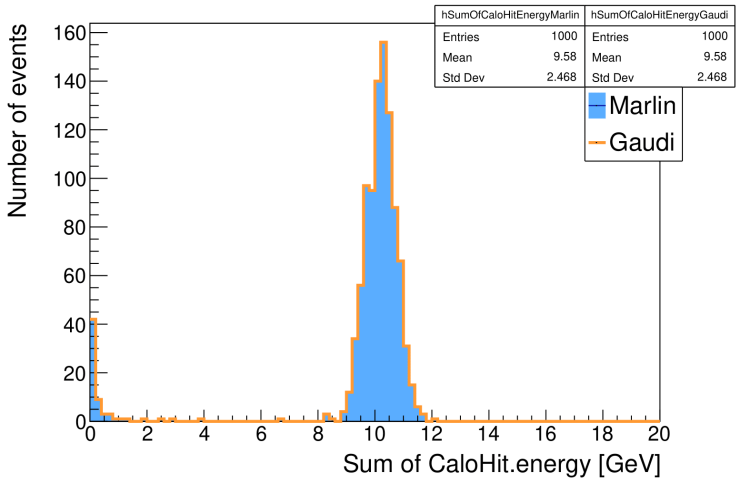



Figure 8: Distribution of the sum of the energy of calorimeter hits from DDCaloDigi.

In summary, this work focused on finalising the porting of the DDCaloDigi algorithm to the Gaudi framework. Following the successful compilation and run of the code, a validation was performed and the results of this validation are presented above. Upon completion of this process, a pull request (PR) for the ported algorithm was opened in the k4GaudiPandora repository. The current status is that the algorithm is nearly ready for merging and integration into the Key4hep software stack.

An additional task in this project involved optimising the code by dividing it into two components: one for the digitization of the electromagnetic calorimeter (ECAL) and the other for the hadronic calorimeter (HCAL). This splitting was successfully implemented and validated. A subsequent idea for further optimisation is to simplify the algorithm by creating a common approach for both ECAL and HCAL, which would depend only on the technology. In principle, a main algorithm could perform the digitisation, with helper functions available to apply digitisation effects to the scintillator or silicon components as necessary. This approach has been started, but it remains under discussion and extends beyond the scope of the current project.

# 4 Conclusion

In this project, the final step for the successful integration of two digitization algorithms, DDSimpleMuonDigi and DDCaloDigi, into the Key4hep framework has been completed. The porting of both algorithms has been finalised and successfully validated. A pull request (PR) for DDSimpleMuonDigi was opened in the k4GaudiPandora repository and has been merged into the Key4hep stack, making it available for use by all Key4hep users. Another PR for DDCaloDigi has been opened in the k4GaudiPandora repository and is nearing completion for merging.

As part of future work, the only algorithm in the DDMarlinPandora package that needs to be ported to Gaudi is DDPandoraPFA. With the porting and validation of it, the entire package will be integrated into Key4hep. This is a work in progress but outside the scope of this summer student project.

Working on this project came with some issues and difficulties, mainly in the porting and compilation step. However, I learned a lot about different frameworks, the structure and principle of work of software for HEP analysis and the digitisation process. During my work this summer, my skills in using C++ and Python programming languages were improved. I was also introduced to using GitHub to use components from the software, track the history of the changes and merge my contribution to the Key4hep software. These skills will be very useful for my future as a scientist.

# Acknowledgements

I would like to thank my supervisors Juan Miguel Carceller and Swathi Sasikumar for their guidance and mentorship throughout this project. Despite the challenges encountered, their insights and support were invaluable in helping me acquire new knowledge and skills.

A special thank you to my fellow summer students and office mates for sharing this experience and making it so enjoyable.

# References

[1] Andre Sailer et al. The key4hep software stack: Beyond future higgs factories, 2023. URL: https://arxiv.org/abs/2312.08151, arXiv:2312.08151.

[2] Erica Brondolin et al. Key4hep: Progress report on integrations, 2023. URL: https://arxiv.org/abs/2312.08152, arXiv:2312.08152.

[3] Linear Collider Software. iLCSoft. https://github.com/iLCSoft, 2024.

[4] Key4hep. k4MarlinWrapper. https://github.com/key4hep/k4MarlinWrapper, 2024.

[5] CERN Linear Collider Detector. CLD - A Detector Concept for the FCC-ee. Technical report, CERN, Geneva, 2019. 75 pages, 67 figures. URL: https://cds.cern.ch/record/2697140, arXiv:1911.12230.