Key4hep and Overlay of Backgrounds

André Sailer

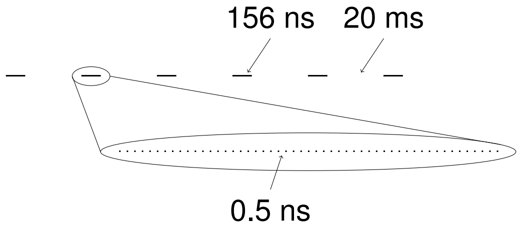
CERN-EP-SFT

ECFA Higgs Factories: 2nd Topical Meeting on Reconstruction
July 11–12, 2023

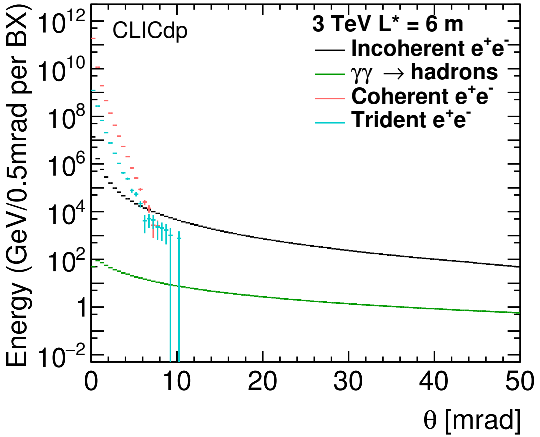
Table of Contents

The Problem (with CLIC as the example)

- The hard physics interaction is accompanied by some kind of "background" processes
  - Incoherent Pairs, Coherent Pairs, $\gamma\gamma \rightarrow$ hadron events, pile-up
- Depending on the beam structure, background event rate, and detector timing realistic reconstruction has to take the deposits from these background particles into account



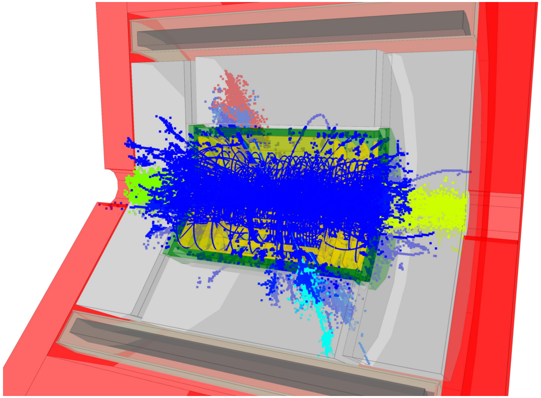
CLIC bunch train structure



Angular distribution of backgrounds for 3 TeV CLIC, detector acceptance starts at 10 mrad

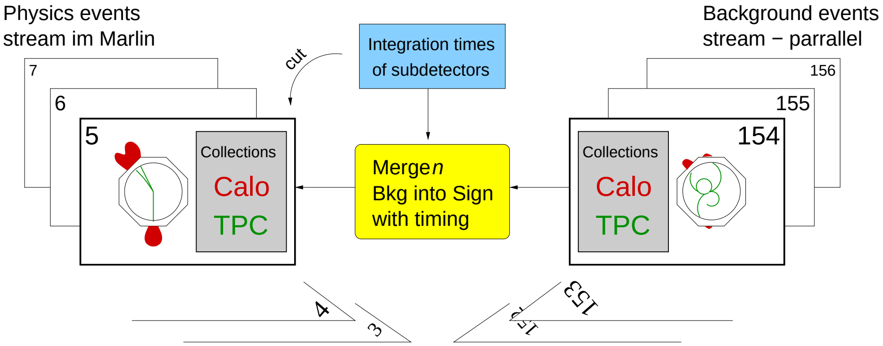
The Approach

- Usually not feasible to (Geant4) simulate complete background for each physics event
    - Simulation bound to specific accelerator parameters
    - Much larger CPU consumption
- Thus simulate background events separate from physics events and *overlay* before digitisation/reconstruction



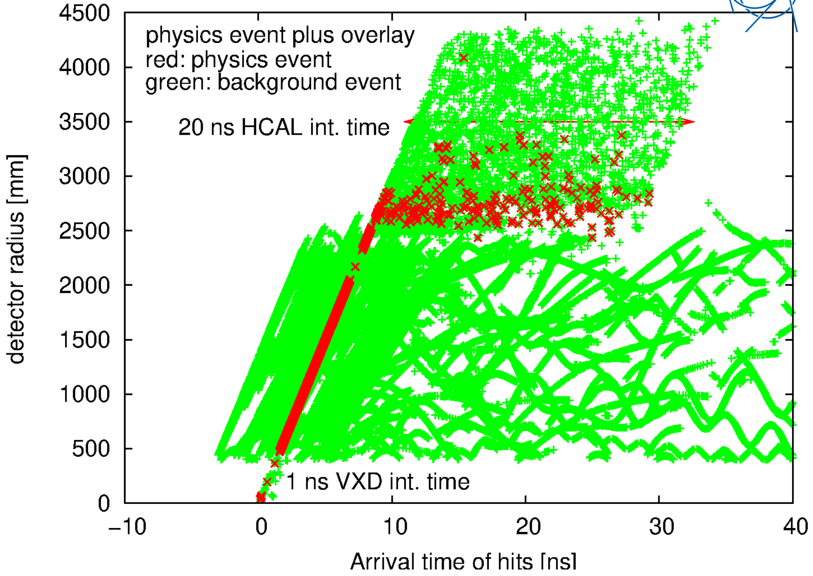
The Processors

- In iLCSoft there are multiple processors to overlay background events to physics events
  - Overlay: Overlay full background events (fixed number, Poisson distributed) to single physics event. Used by ILD
  - OverlayTimingGeneric [1]: Overlay background events with shifting T_0 to account for background from out-of-time events, apply timing window of detectors
- Multiple Overlay processors can be used sequentially to overlay different background sources with different rates.



OverlayTiming(Generic)

- Example output for overlaid event of CLIC_ILD_CDR (with a TPC main Tracker)
- Physics at $t = 0$ ns, some background events before, more after



Overlay Background at Scale

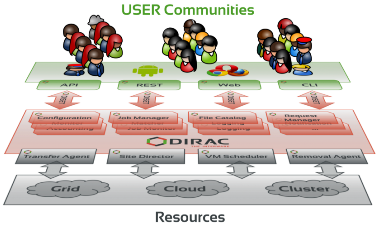
Now that we are able to overlay background to physics events. How do we do this for a couple million physics events or more?

- We simulate a *sufficient* number of background events
- Make use of Storage and Computing Resources around the World
- With iLCDirac

(iLC)Dirac in a Nutshell

iLCDirac [2] is based on the DIRAC interware originally developed for LHCb[3]

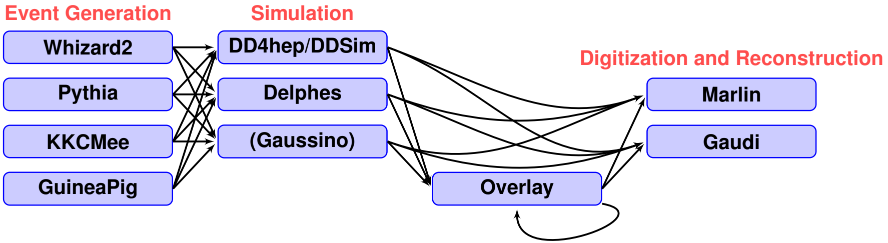
- Dirac (Distributed Infrastructure with Remote Agent Control): High level interface between users and distributed resources
- Distributed Workload Management: one interface to execute anywhere: batch farms, grid computing elements, HPCs
- Data Management (file transfers, meta data augmented file catalog)
- High degree of automation
- Web interface for controlling jobs



diracgrid.org

iLCDirac Use Cases

- The iLCDirac extension of Dirac is set up for the ILC, Calice, and FCC Virtual Organisations
- Centralized MC Production (Event Generation, Geant4 Simulation, Reconstruction)
- User jobs (Generation, Simulation, Reconstruction, Analyses)
- iLCDirac uses almost all functionality provided by DIRAC
- Specific features in iLCDirac
  - Workflow Modules for Key4hep Software

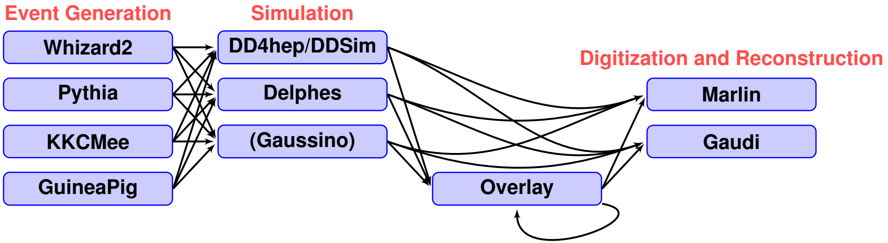


  - Overlay System for adding beam background files to MC jobs efficiently and effectively
  - Pandora Particle Flow calibration service (Marlin based)

Source Code: **https://gitlab.cern.ch/CLICdp/ILCDIRAC**

iLCDirac Use Cases

- The iLCDirac extension of Dirac is set up for the ILC, Calice, and FCC Virtual Organisations
- Centralized MC Production (Event Generation, Geant4 Simulation, Reconstruction)
- User jobs (Generation, Simulation, Reconstruction, Analyses)
- iLCDirac uses almost all functionality provided by DIRAC
- Specific features in iLCDirac
  - Workflow Modules for Key4hep Software



  - **Overlay System for adding beam background files to MC jobs efficiently and effectively**
  - Pandora Particle Flow calibration service (Marlin based)

Source Code: **https://gitlab.cern.ch/CLICdp/ILCDIRAC**

User Job with Reconstruction

```
from DIRAC.Core.Base import Script
Script.parseCommandLine()
from ILCDIRAC.Interfaces.API.DiracILC import DiracILC
from ILCDIRAC.Interfaces.API.NewInterface.UserJob import UserJob
from ILCDIRAC.Interfaces.API.NewInterface.Applications import Marlin
dIlc = DiracILC()
job = UserJob()
job.setInputData( "/ilc/prod/clic/350gev/h_nunu/ILD/SIM/00006524/000/h_nunu_sim_6524_1.slcio" )
job.setOutputData( "myReco_1.slcio" )

marlin = Marlin()
marlin.setVersion( "v0111Prod" )
marlin.setInputFile( "h_nunu_sim_6524_1.slcio" )
marlin.setOutputFile( "myReco_1.slcio" )
marlin.setSteeringFileVersion( "V22" )
marlin.setGearFile( "clic_ild_cdr500.gear" )
marlin.setNumberOfEvents( 10 )


marlin.setSteeringFile( "clic_ild_cdr500_steering_350.0.xml" )


job.append( marlin )
job.submit(dIlc)
```

User Job with Overlay

```
from DIRAC.Core.Base import Script
Script.parseCommandLine()
from ILCDIRAC.Interfaces.API.DiracILC import DiracILC
from ILCDIRAC.Interfaces.API.NewInterface.UserJob import UserJob
from ILCDIRAC.Interfaces.API.NewInterface.Applications import Marlin, OverlayInput
dIlc = DiracILC()
job = UserJob()
job.setInputData( "/ilc/prod/clic/350gev/h_nunu/ILD/SIM/00006524/000/h_nunu_sim_6524_1.slcio" )
job.setOutputData( "myReco_1.slcio" )

over = OverlayInput()
over.setBXOverlay( 300 )
over.setGGToHadInt( 0.0464 )
over.setNumberOfSignalEventsPerJob( 100 )
over.setBackgroundType( "gghad" )
over.setDetectorModel( "CLIC_ILD_CDR500" )
over.setEnergy( "350" )
over.setMachine( "clic_cdr" )
# Marlin as before, except
marlin.setSteeringFile( "clic_ild_cdr500_steering_overlay_350.0.xml" )

job.append( over )
job.append( marlin )
job.submit(dIlc)
```

Selection of Background Files

When the job runs, for each requested background overlay, the system

- calculates how many background files are needed, up to a configurable limit
- looks up the list of files in the iLCDirac FileCatalog, including information where files are available
- randomly selects some of them
- downloads the files to the job's scratch area, preferring closer Storage Elements

Integration to the Steering File

With the background files locally available

- the steering file is modified to assign the files to the respective processor/algorithm
- adapt parameters to those set before

And then the reconstruction runs

Further Thoughts On the Overlay System

- The original design goal of the overlay system in iLCDirac had the goal to avoid overloading Storage Elements with too many parallel requests
- Total number of parallel downloads of background files is limited through a central service, and staggered in time
- Other limitations can be set through DIRAC features itself

Conclusions

- Sufficient overlay functionality for ILC, CLIC, and FCCee Studies exists in the iLCSoft software and computing environment
- Background overlay at scale is working efficiently
- Overlay processors still need to be implemented for EDM4hep
- Distribution and use of background files with iLCDirac possible also for EDM4hep
- Hooks into the Gaudi steering files to be implemented once algorithm parameters clear

Acknowledgments

This work benefited from support by the CERN Strategic R&D Programme on Technologies for Future Experiments **(CERN-OPEN-2018-006)**.
This project has received funding from the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement no. 871072.

References

[1] P. Schade and A. Lucaci-Timoce. *Description of the signal and background event mixing as implemented in the Marlin processor OverlayTiming*. CERN LCD-Note-2011-006. 2011.

[2] Christian Grefe et al. “ILCDIRAC, a DIRAC extension for the Linear Collider community”. In: *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013)*. Vol. 513. CLICdp-Conf-2013-003. Amsterdam, the Netherlands, 2013, p. 032077. DOI: `10.1088/1742-6596/513/3/032077`.

[3] A Tsaregorodtsev et al. “DIRAC: a community grid solution”. In: *J. Phys.: Conf. Ser.* 119 (2008), p. 062048.