CMS Monte Carlo Generators: Run II experience and needs for Run III

Luca Perrozzi (ETH Zurich) for the CMS collaboration

CHEP conference
Sofia, July 9th 2018
Introduction

- CMS at LHC relies on detailed and large scale Monte Carlo production for modeling of the detector and underlying physics

- We are hitting several bottlenecks in term of cpu and disk space: robust software and computing infrastructure more and more essential to reach the physics goals

- Typical Monte Carlo workflow has a few distinct steps
  - Hard process/Matrix Element generation: Generate kinematic configuration for a desired process up to parton level using perturbative QCD
  - Parton Shower/Hadronization: Adds additional QCD and QED emissions down to a low scale, and produces hadrons from QCD partons
  - Detector Simulation(*) and Digitization: Detailed Geant4 simulation of the interactions of the outgoing particles with the CMS detector, followed by simulation of detector electronics and creation of simulated raw data
  - Reconstruction: Reconstruction of simulated raw data into higher level physics objects
    - To a good approximation, identical code as runs on real data

- Strong motivation for NLO and/or multi-leg/merged-multiplicity Monte Carlo generators whenever possible
  - Achieve highest accuracy for final states with additional jets
  - Warning: large time/event required
  - Warning: NLO features negative weights, requiring up to x10 statistics wrt LO

(*)cfr Talk “Current and Future Performance of the CMS Simulation” by K. Pedro
CMS Software Overview

• **Main CMS software application: CMSSW**
  – Modular C++ application which can be used for event generation, detector simulation, reconstruction, and analysis

• **Configuration of CMSSW runs steered with python files**

• **Input and output through ROOT-based Event Data Model (EDM) files**
  – Storing run-level, lumi-section-level (23s periods for real data), or event-level data products

• **CMSSW links directly to many externals**
  – Externally maintained C, C++, fortran, or python software which is either an indirect dependency or is directly called from within CMSSW
  – Externals are compiled with the same common libraries, compiler version as CMSSW and packaged together with a given release
  – Starting from either a tarball from the author’s website, from GENSER, or from a cms-managed github mirror

• **Rapidly evolving software**
  – Swift change of compilers, OS versions, multi-thread support ...
  – Difficult for the ‘externals’ to keep up
CMS Submission Infrastructure Overview

• Large-scale submission of CMSSW jobs to grid resources managed with python-based tools
  – For central production of Monte Carlo, data processing, etc

• Jobs are assumed to run through CMSSW
  – Configured by the corresponding python-based file

• All inputs and outputs are assumed to be EDM files (with a few special cases)

• CMSSW software and corresponding externals are made available on worker nodes through CVMFS
  – Distributes http-based read-only filesystem

• CMS software evolution and multi-threading pose severe challenges to old/legacy MC software
  – CMS has to produce MC events compatible to each collected dataset, even after many years the data was taken
  – Often ad-hoc “features” needed, difficult to maintain

(*)cfr CHEP 2016 talk “Software and Experience with Managing Workflows for the Computing Operation of the CMS Experiment” by J.-R. Vlimant
• **Basic paradigm for event generation**
  - C++ module making calls to linked external generator code to produce, for each event, HepMC::GenEvent to be stored as EDM

• **Matrix element generators produce LHE files** (Madgraph, POWHEG, ...)
  - Loosely coupled to CMSSW, calling of external generation script handled by CMSSW module “externalLHEProducer”
  - ASCII LHE files are transient and immediately packed into binary compressed format

• **Parton shower programs finalize the full event properties** (Pythia, Herwig, Sherpa)
  - Fully integrated as “externals” and packaged or linked directly from CMSSW interfaces
Gridpacks

- **Matrix element generators typically have two discrete steps:**
  - Matrix element/code generation and phase space integration
  - Generation of events

- **Compiled code and results of the phase space generation stored into gridpacks**
  - For efficient generation of events on the grid
  - Largely self-contained tarballs prepared in advance and stored on CVMFS
  - Modeled on the built-in generator functionalities or through dedicated scripts
    - Maintainence of scripts for all the major generators (MG5 aMC, POWHEG, Sherpa, etc) can be quite heavy
  - Gridpack production is the default and most widely supported mechanism in CMS for Run 2

- **Gridpack preparation done exploiting batch-job parallelism (LSF or Condor)**
  - Compiling code on batch workers and long init time for event generation discouraged
  - Gridpack size can be an issue (>500MB for the tarball and 5GB decompressed)
  - Gridpack generation step needs reliability and reasonable run-time “as the physicist waits”
  - Recent and important developments through HTCondor Global Pool infrastructure(*)

- **Grid jobs using gridpacks from CVMFS can generate events with trivial process-level parallelization**

(*)cfr Poster “Producing Madgraph5_aMC@NLO gridpacks and using TensorFlow GPU resources in the CMS HTCondor Global Pool” by K.H. Anampa
Evolution of generators and tunes

- Major changes in generator releases, PDF versions and generator tunes for 2017 data
  - New tunes (GEN-17-001, in preparation) and PDF (NNPDF3.1) derived to include 13 TeV data
    - Previous major update in 2014, based on 7 TeV data
  - Better Underlying event and jet multiplicity description

- Such updates require careful planning and coordinated effort lasting several months
  - Careful validation of generators and production setups
  - Large scale production of $O(10K)$ gridpacks

- Next major goal: coherent MC production with same settings for the full Run 2
LHC Run2 experience in a nutshell

- **Computing resources planned ~1.5-2y in advance**
  - Current CMS MC budget is $O(10-15B)$ events per year, $O(10K)$ different requests
  - Competing resources with data taking, re-processing, upgrade and validation samples
  - Budgets do NOT scale linearly with LHC performance and effective integrated luminosity...

- **Start to plan each MC production campaigns $O(6)$ months in advance**
  - Generator configurations and tunes, software version, reco conditions and corrections, production workflows need to be fully validated under time pressure

- **Moving more and more from fully inclusive datasets to fully exclusive datasets**
  - We'll have to make this work with more efficient slicing and weighting
  - Switching from search to precision: higher precision (NLO) requires higher resources...

MC for 2016 data
More than 20B events produced

MC for 2017 data
Close to 6B events produced so far, as much in the queue
Towards Run3 and beyond: it will only get worse
MC priorities and budgets from now to Run3

• MC planning driven by physics priorities

• Physics priorities driven by machine conditions
  – No major center of mass energy jumps
    • 13 to 14 TeV will not be a dramatic increase as from 8 to 13 TeV
  – Once “bulk of the distributions” explored, focus on “tails” of phase space
    • Notable exceptions are ultimate precision SM measurements
      – e.g. Weak mixing angle, top mass, W mass

• Need to “fight” against conflicting requirements:
  – (Much) larger datasets
  – Increased measurement precision
  – Need for alternative samples to estimate systematics uncertainties
    • Different generators or parameters
    – Flattening of computing resources (both cpu and disk space)

• Need to find an evolution “model” that scales already for end of Run2
  – Run2 is collecting >5 times data wrt Run1
  – Favor cpu (i.e. don’t store LHE, “cheap” to reproduce) wrt disk occupancy
  – Store slimmed dataformats like MINIAOD and NANOAOD(*)

(*)cfr Talk “A further reduction in CMS event data for analysis: the NANOAOD format” by A. Rizzi
Generator technical developments: desiderata

- **A vast program of MC software improvement must be pursued in the next 5 years**
  - This can’t be left on the shoulders of theorists and MC builders, and must be tackled community-wide
  - A community white paper has been issued to illustrate possible avenues (*)

- **Examples of needed technical improvements:**
  - Faster phase space integration
  - Drastic reduction of fraction of events with negative weights for NLO precision
    - E.g. folding of the integration phase space implemented in POWHEG
  - Drastic increase of the matching efficiency (currently ~30%)
  - Continue pursuing reduction of memory consumption to match higher jets and parallelization
    - Currently up to 4j at LO, up to 2j at NLO (expected more with low memory multicore option)
  - Complete integration of process independent NLO QCD x EWK corrections
    - Up to high multiplicity final states, for both virtual and real contributions
    - Properly interfaced to parton shower
  - Bias weights for both LO and NLO
    - Produce inclusive samples with enriched statistics for specific phase spaces
  - Large flexibility for LHE level cuts for both LO and NLO
    - HT, VpT, number of additional jets, VBF-like, etc
  - Massive use of events weights for systematic uncertainties
    - Routinely used in Matrix Elements for PDF and perturbative QCD scales
    - Recent Pythia8 versions support weights for parton shower systematics
  - Single-gridpack parameter scan
    - Especially for BSM scans, so far very high number of gridpacks required

(*) cfr Talk “The HEP Software Foundation Community White Paper” by M. Jouvin
Conclusions

• **Monte Carlo Generators** are an essential aspect of the Physics program of every experiment
  – LHC is not an exception

• **Robust software and computing infrastructure** essential to satisfy physics needs and goals
  – Several challenges posed by the large amount of data collected and long timescale required for the MC to be available/reproducible

• **Need to find evolution “models”** that scale for the future

• **Experiments need to work in close contact with MC authors** to improve program performance, precision and flexibility
  – Community white paper issued recently
Backup slides
Multi-leg generators and gridpacks

- Several Monte Carlo generators have the capability to automatically generate matrix elements at LO for several jet multiplicities.

- Results can be consistently combined across multiplicities when treated consistently in parton shower.

- Most widely used configuration in CMS Run 2: Madgraph aMC@NLO (LO) + Pythia 8 with MLM matching.

- Most complex processes with up to 4 additional jets.

- CPU time up to about 10s per matrix element event (averaged over jet multiplicities), with O(10%) matching efficiency at the parton shower step → 100s cpu for matrix element per fully-simulated event.
NLO Multi-leg generators

• A few Monte Carlo generators now have the capability to (semi)-automatically generate matrix elements at NLO in QCD for several jet multiplicities and consistently merge them

• Most widely used configuration in CMS Run2: Madgraph aMC@NLO (NLO) + Pythia 8 with FXFX merging

• At NLO in QCD, each multiplicity consists of born, real, and virtual contributions to the matrix element

• Most complex processes with up to two additional jets at NLO

• CPU time scaling up to to ~ 30 s per ME event with matching efficiencies of ~ 30% → 90 s of cpu time for matrix element per fully-simulated event

• Events are also accompanied by a possibly large fraction of negative weights (up to 40%) which reduces statistical precision and necessitates larger samples

• Diagram/code generation also very CPU and memory intensive → recent contribution to 2.4.x series to significantly improve (thread-level) parallelization and memory footprint of this step, eventually enabling more complex processes
import FWCore.ParameterSet.Config as cms

from Configuration_GENERATOR.Pythia8CommonSettings_cfi import *
from Configuration_GENERATOR.Pythia8CUEP8M1Settings_cfi import *

generator = cms.EDFilter("Pythia8GeneratorFilter",
    maxEventsToPrint = cms.untracked.int32(1),
    pythiaPylistVerbosity = cms.untracked.int32(1),
    filterEfficiency = cms.untracked.double(1.0),
    pythiaHepMCVerbosity = cms.untracked.bool(False),
    comEnergy = cms.double(13000.0),

    crossSection = cms.untracked.double(1.92043e+07),

    PythiaParameters = cms.PSet(
        pythia8CommonSettingsBlock,
        pythia8CUEP8M1SettingsBlock,
        processParameters = cms.vstring(
            'HardQCD:all = on',
            'PhaseSpace:pTHatMin = 50 ',
            'PhaseSpace:pTHatMax = 80 ',
        ),
        parameterSets = cms.vstring('pythia8CommonSettings',
            'pythia8CUEP8M1Settings',
            'processParameters',
        )
    )
)
CMS Software: LHE Input

• CMS maintains its own LHE parser (based on xerces-c xml library)

• An LHE file can be read as input to a CMSSW job and is converted on the fly to C++ classes LHERunInfoProduct and LHEEventInfoProduct which store the relevant information and can be stored/read from EDM files (support for per-event weights added to CMS lhe parser and classes)

• LHE information can be passed as input to a hadronizer as part of the event generation step in CMSSW (using for example the Pythia8::LHAup mechanism to pass the needed information on the fly in memory)

• LHE parsers included with Pythia, Herwig etc are not used
  – Advantage: Uniform hadronizer-independent storage and access to lhe information
  – Disadvantage: We have to maintain our own LHE parser

• CMS production tools do not work transparently with ascii LHE input (metadata not automatically available in data management system, skipping of events is inefficient, etc)

• It is possible to use privately produced LHE files for central production (user copies the files to eos and then a conversion step is run to produce EDM files containing the LHE products, which can then be used for further production steps for hadronization, simulation, etc)

• Disk space, file corruption, etc, are major issues when dealing with large sets of LHE files in this way
Parameter Scans

• Recently added some functionality in CMSSW for parameter scans (used so far for SUSY signal MC)
• SUSY signal production with MG5 aMC@NLO + Pythia 8 (LO MLM)
• Typical case: gluino/squark pair production (+0,1,2 jets LO) in MG5 aMC@NLO, decay in Pythia, steered by SLHA table
• Gridpack and pythia configuration+SLHA table for decays are randomly selected for each luminosity section (≈200 events after matching)
• Resulting sample contains a mixture of all scan points
• High granularity of randomization ensures missing events from job failures are randomly and ≈ eveny distributed across scan points