Generators Integration into CMS Software and Production

Josh Bendavid (Caltech)
for the CMS Generators group

Caltech

January 11, 2016
ATLAS-CMS Monte Carlo Generators Workshop

Introduction

- Important for the theory community, and especially generator authors to understand **how** we are using external generator programs in CMS at both a physics and technical level
- Better understanding of our needs and workflows $\rightarrow$ more effective use of generators in CMS, suggestions for how we can improve on what we are doing

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 is steered with python-based configuration files
- Input and output with root-based EDM files, which store 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

CMS Production Overview

- Python-based tools manage large-scale submission of CMSSW jobs to grid resources for central production of Monte Carlo, data processing, etc
- Jobs are assumed to be CMSSW jobs configured by the corresponding python-based configuration
- All input and output are assumed to be EDM files (with a few special cases)
- A similar mechanism is available to end users to submit analysis jobs
- CMSSW software and corresponding externals is made available on worker nodes through CVMFS (distributes http-based read-only filesystem)

CMS Software: Event Generation

- Basic paradigm: A C++ module with a common interface makes the needed calls to a linked external generator code in order to produce for each event a HepMC::GenEvent, which can then directly stored in the EDM output
- Configuration of the generator takes place within the CMSSW python configuration
- **Advantages:**
  - Uniform configuration and IO mechanism (production tools only have to deal with CMSSW)
  - No intermediate files needed (HepMC::GenEvent is passed along in memory to standard CMSSW/root IO mechanisms or directly to GEANT, which is also called from inside CMSSW)
- **Disadvantages:**
  - Each generator needs a dedicated interface needed in CMSSW and needs to be packaged as a CMSSW external
  - Initialization and event generation calls must be possible from within a C++ application
- In practice, Pythia, Herwig, Sherpa fit very nicely into this paradigm (with some preference for C++-based versions)

Example CMSSW GEN Configuration Fragment

```
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

LHE Input for Central Production

- 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

Central production of LHE events

- LHE generators like Madgraph_aMC@NLO, POWHEG, etc cannot be directly called from within CMSSW in general
- Solution is "externalLHEProducer" a C++ CMSSW module which calls an external script, then reads the resulting LHE file (with the CMS lhe parser) and produces the necessary LHERunInfoProduct/LHEEventInfoProduct which can be stored in the EDM file and/or passed along to the hadronizer
- Further issue: LHE generator code cannot easily be included with CMSSW as an external, since each process requires dedicated (and sometimes dynamically generated) libraries
- Solution: "gridpacks" with pre-generated/compiled code, and with initial phase space integration results stored in a tarball
- Gridpacks are put in CVMFS and can be accessed by jobs (gridpack location is a configuration parameter of the externalLHEProducer module)
- Minimal and compact external input, and compressed EDM output make very large scale LHE production possible.
- CMS has produced 27 billion LHE events through this mechanism for Run 2 so far.

Gridpacks

- General gridpack mechanism used in CMS is modeled on the built-in functionality for LO processes in Madgraph_aMC@NLO
- We maintain scripts for Madgraph_aMC@NLO (including NLO processes), POWHEG, JHUGen to produce gridpack tarballs based on the appropriate input cards
- Important considerations:
  - Compiling code on batch workers is discouraged (should be possible to fully precompile everything)
  - Long initialization time for event generation is discouraged
  - Gridpack size is an issue (more than about 500MB for the tarball or 5GB decompressed starts to become problematic)
  - (For Madgraph_aMC@NLO we use lzma compression with very large dictionaries because of large use of space from duplicated code in statically linked executables for each subprocess)
  - Gridpack generation step needs reliability and reasonable run-time "as the physicist waits" (we can use multi-core machines and/or condor/lsf batch queues to do the phase space integration, but does no good if process is bottle-necked by single-threaded steps, or individual long-running jobs)

Parameter Scans

- CMS workflows not very well suited for parameter scans
- Currently two possibilities:
  - One configuration and/or gridpack and one Monte Carlo sample for each parameter point (lots of book-keeping and preparation work)
  - Privately produce lhe files for each parameter point and mix them together in a single eos directory to be imported into CMS production (unwieldly production and handling of lhe files)
- Some thought and effort towards improving the situation. For integrated generators like Pythia, Herwig, Sherpa could randomize configuration and re-initialize every N events for example
- Solution is also needed on the LHE generation side (ideally without requiring one gridpack per parameter point)

Source Code Availability

- CMSSW and all of its dependencies and externals (and their dependencies) are open-source
- Practical requirement: Externals (and any code included in gridpacks) need to be compiled with compatible compilers/libraries/etc to link to CMSSW and/or run within a CMSSW environment on the grid
- Further Benefit: Substantial manpower and expertise within the experimental collaborations. We are happy to help debug issues with the software we are using. Source code (and publicly accessible cvs/svn/git repository) make it much easier for us to do this
- Further consideration: We should be able to know exactly what we are putting into the Monte Carlo samples used for our papers

Patching of Generators

- We strongly prefer not to apply our own patches to generator code for obvious reasons of reproducibility and communication outside CMS
- Release schedules for generator tools don't always line up with our own production campaign schedules, so sometimes patches are necessary (but we always discuss it with the authors first at least)
- Very long gap between releases can particularly make this an issue

Schedule Reminder

- Lead-time to produce billions of fully simulated and reconstructed Monte Carlo events is long
- Pythia-based production for Run 2 started in October 2014
- LHE-based production started (late) in Feb. 2015
- Lots of inertia to change things like tunes, pdf's, etc

Conclusions

- Generator tools need to be carefully integrated into our software and production framework to be useful for large-scale usage
- This process is driven by important constraints on both the CMS side and in the generator tools
- Discussing these issues and technical details can hopefully lead to better and more effectively used generator tools and interfaces