Integration of New Simulation Technologies in the Experiments

Kevin Pedro (FNAL)
on behalf of ATLAS, CMS, LHCb
Joint HSF/OSG/WLCG Workshop
March 21, 2019
• HL-LHC presents significant computing challenges
  ⊗ 10× data vs. Run 2/3
  ⊗ >10× increase in # channels for upgraded detectors
  ⊗ May need improved physics models to simulate complex detectors accurately
  ⊗ Reconstruction will take a larger fraction of CPU time (200PU vs. ~30PU in Run 2)

 Detector simulations need to produce more events with more accuracy in a more complicated geometry… w/ smaller fraction of total CPU usage

Possible solutions: vectorized software and heterogeneous hardware
Vectorized Components

• VecCore ([GitHub/root-project](https://github/root-project))
  - Libraries: Vc, UME::SIMD, CUDA

• VecMath ([GitHub/root-project](https://github/root-project))
  - Spinoff of VecCore, dedicated to math utilities (including RNGs)

• VecGeom ([CERN/GitLab](https://cern/gitlab))
  - Vectorized geometry and navigation, multi-particle interface

• GeantV ([CERN/GitLab](https://cern/gitlab))
  - Vectorized transport engine for detector simulation: geometry navigation, magnetic field, EM physics
  - [pre-beta tags](https://github/osgt) now available, beta release coming soon
DD4hep in the Experiments

• Not a vectorized package, but a common solution for detector description
• Uses ROOT TGeo to handle geometry construction

CMS:
• Infrastructure for migration in place, tested w/ muon system (drift tubes)
• Provided significant feedback to DD4hep developers
• Contacts established to migrate other subdetectors

ATLAS:
• Testing DD4hep as a description language
• Need to use GeoModel for backend rather than TGeo (not supported)

LHCb:
• Testing w/ Gaussino, new lightweight simulation framework (CHEP2018)
• Provided feedback for TGeo (optical surfaces)
• Plan to use DDG4 simulation toolkit to convert geometry for Geant4
VecGeom in the Experiments

CMS:
• VecGeom used in scalar mode with Geant4
• 7–13% speedup with similar memory usage
  → Just from code improvements, no vectorization!
➢ Included in production for >1 year

ATLAS:
• VecGeom tested in scalar mode with Geant4: just Cones and PolyCones used
• 1–3% speedup observed
• Testing with all shapes from VecGeom in progress
  o Also testing w/ different Geant4 versions (10.4.2, 10.5)

LHCb:
• VecGeom tested in scalar mode with Geant4
• No speedup observed: likely due to simple shapes in detector geometry
GeantV in the Experiments

CMS:
- Significant effort in testing
  - First stage: cmsToyGV (in GeantV alpha)
    - Integrate w/ simplified toy-rt-framework
    - Demonstrate compatible threading model
    - Example of ExternalLoop mode
  - Second stage: integration in CMSSW
    - See SimGVCore
    - In progress, details to follow
  - Third stage: performance testing
    - Use beta release when available

LHCb:
- Plan to integrate w/ Gaussino
Elements of GeantV Integration in CMSSW

- Generate events in CMSSW framework, convert HepMC to GeantV format
- Build CMSSW geometry natively and pass to GeantV engine (using TGeo)
  - Using constant magnetic field, limited EM-only physics list
  - Adaptation of sensitive detectors and scoring in progress
- Up to date with latest tag pre-beta-4
- Run GeantV in “external loop” mode using CMSSW ExternalWork feature:
  - Asynchronous task-based processing

![Diagram showing external processing and CMSSW module interaction]
Geant4 vs. GeantV Scoring

- **Sensitive detectors (SD) and scoring** trickiest to adapt
  - Necessary to test “full chain” (simulation → digitization → reconstruction)
  - Significantly more complicated than Geant4 MT

Geant4 shares memory, but each event processed in separate thread

- Duplicate SD objects per event per thread, then aggregate
  - → 4 streams = 16 SD objects
  - GeantV TaskData supports this approach
Technical Details of Scoring

• SD code uses Geant4 objects heavily: G4Step, G4LogicalVolume, etc.
• GeantV instead has Geant::Track, vecgeom::LogicalVolume, etc.
  ➢ Need “Rosetta stone” to translate various operations
    o Geometry volumes and replica numbers, material properties, etc.
    o Thanks to GeantV developers for sending this information
• How to avoid upsetting delicate and complicated SD code?
  ➢ Template wrappers to unify interfaces and operations
    o Store pointers or references to original objects – minimize memory overhead
    o No extra cost from virtual table (vs. using inheritance)
• Implementation in progress using simplified SD class
  o Future: connect to TaskData, then migrate real SD classes
ATLAS FastCaloSim on GPU

• Self-contained simulation → feasible target to profile and explore GPU port
• Standalone version runs in ROOT interpreter
  ➢ Working on compiled version (easier profiling and porting/parallelization)
• Identified possible parallelization for
  TFCSLateralShapeParametrizationHitChain::simulate()
  o Implementing CUDA kernel & device functions
  o Try to minimize data transfer between CPU and GPU
• Next steps:
  o Optimize CUDA functions
  o Apply CUDA stream
  o Use many GPU devices in same node (e.g. Summit) or distributed nodes
• Funded by HEP-CCE
• Collaboration between ATLAS and BNL Computational Science Initiative
Conclusions

• New packages relevant to simulation are available:
  o Common solutions (DD4hep)
  o Vectorized components (VecGeom, GeantV, etc.)
• Experiments are making progress testing and integrating these packages
  o Providing frequent feedback to developers
  o Continued communication is essential to the success of these projects
  o Observed speedups vary; many factors at play, and still early
• In particular, CMS integration testing of GeantV is maturing
  o Next step: performance testing w/ beta release
    ➢ Check if speedup translates to experimental software framework
    ➢ Check if existing CMS speedups are compatible w/ GeantV
    ➢ Understand full cost of migration to new interfaces
  o Provide a path for other experiments to follow
Backup
External Work in CMSSW (1)

Setup:

• TBB controls running modules
• Concurrent processing of multiple events
• Separate helper thread to control external
• Can wait until enough work is buffered before running external process
External Work in CMSSW (2)

Acquire:
- Module *acquire()* method called
- Pulls data from event
- Copies data to buffer
- Buffer includes callback to start next phase of module running
External Work in CMSSW (3)

Work starts:

• External process runs
• Data pulled from buffer
• Next waiting modules can run (concurrently)
External Work in CMSSW (4)

Work finishes:

• Results copied to buffer
• Callback puts module back into queue
External Work in CMSSW (5)

Produce:

- Module `produce()` method is called
- Pulls results from buffer
- Data used to create objects to put into event
FastCaloSim Profile

- **TFCSLateralShapeParametrizationHitChain::simulate()** is the most significant routine except I/O parts.
- The running time of **TFCSLateralShapeParametrizationHitChain::simulate()** will increase as the number of hits increases.
- **TFCSLateralShapeParametrizationHitChain::simulate()** is a possible candidate to parallelize.
TFCSLateralShapeParametrizationHitChain::simulate() Structure

TFCSLateralShapeParametrizationHitChain::simulate() {
    ...
    ...
    Loop on \texttt{nhit}
    ...
    Loop on \texttt{hit\_simulation\_chain}
    Call
    End Loop
    Possible Parallelization
    ...
    End Loop
    ...
    }

Impossible Parallelization because of data dependency