Status of GeantV Integration in CMSSW

Kevin Pedro, Sunanda Banerjee
(FNAL)
October 4, 2019

Introduction

- Integration testing of GeantV w/ CMSSW has several goals:
  - Demonstrate benefits of co-development between R&D team & experiments
  - Exercise capabilities of CMSSW framework to interface with external processing (ExternalWork mechanism) and handle track-level parallelization in detector simulation
  - Measure any potential CPU penalties when running GeantV in CMSSW
  - Estimate cost of adapting to new interfaces and eventually migrating to new (and potentially backward-incompatible) tools such as GeantV
    - Thinking forward to HPC/GPU solutions
- *Not* planning to migrate CMS simulation to GeantV
  - Geant4 collaboration has not endorsed the project
  - This is an R&D exercise

GeantV Integration Tests in CMSSW

- Repositories: install-geant, SimGVCore

✓ Generate events in CMSSW framework, convert HepMC to GeantV format

✓ Build CMSSW geometry natively and pass to GeantV engine (using TGeo)

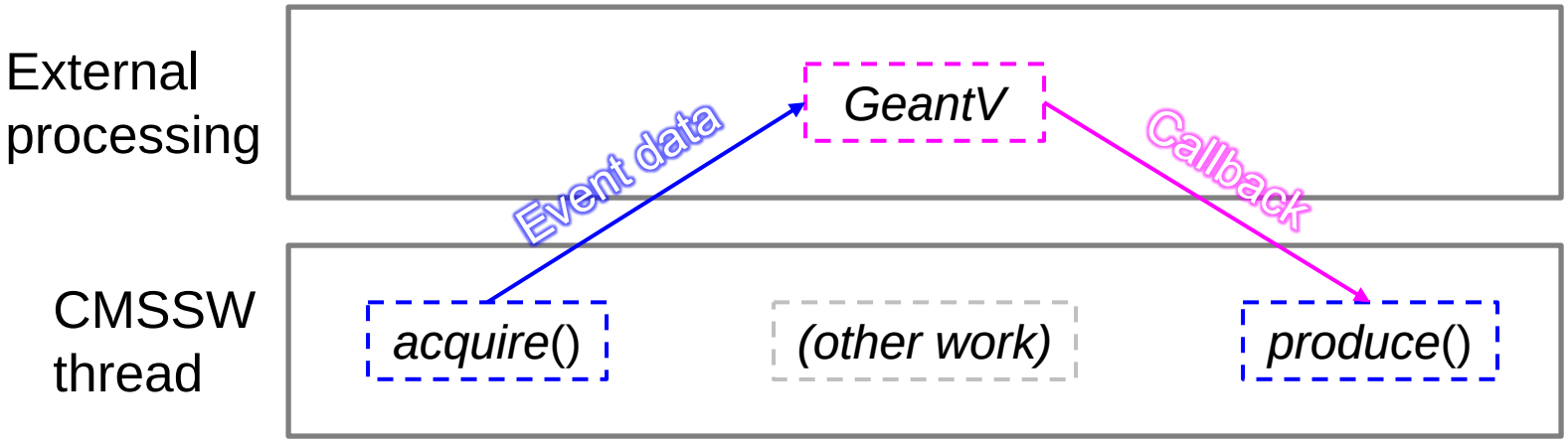
- Using constant magnetic field, limited EM-only physics list

✓ Calorimeter scoring adapted

✓ Run GeantV using CMSSW ExternalWork feature:

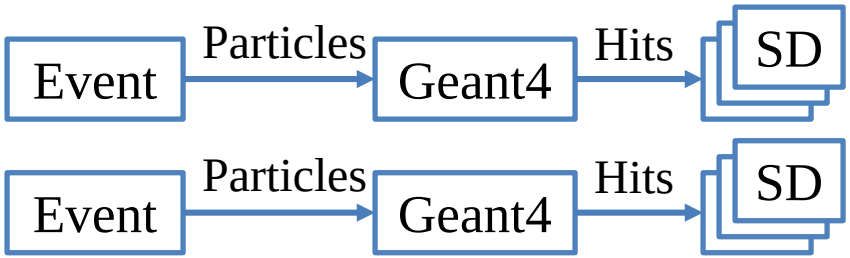
  o Asynchronous, non-blocking, task-based processing

✓ Output in CMS format, immediately suitable for digitization etc.

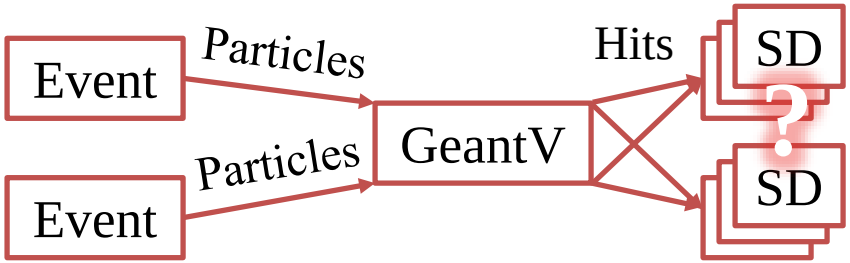


Geant4 vs. GeantV Scoring

- 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

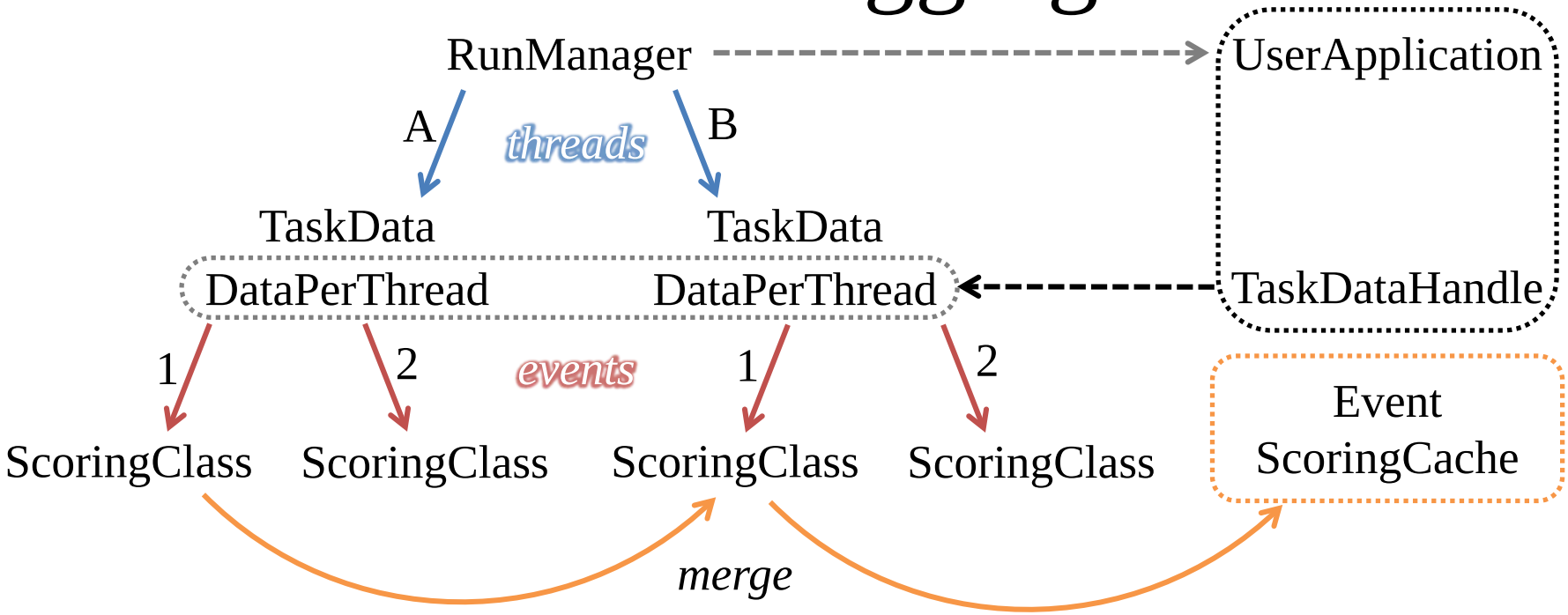


Each event processed in multiple threads, mixed in with other events

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

➢ Use template wrappers to unify interfaces and operations
  - Ensure exact same SD code used for Geant4 & GeantV
  - Minimize overhead (no branching or virtual table)

GeantV Data Aggregation



- Each ScoringClass object has instance of CaloSteppingAction
  - Some additional memory overhead from duplicated class members
    - Attempt to minimize this by storing volume maps in magic static struct
- Merged ScoringClass output copied to cache attached to Event object
  - GeantV may consider event finished before CMSSW has written output → copy to cache, then immediately clear ScoringClass objects (avoid possible race conditions)

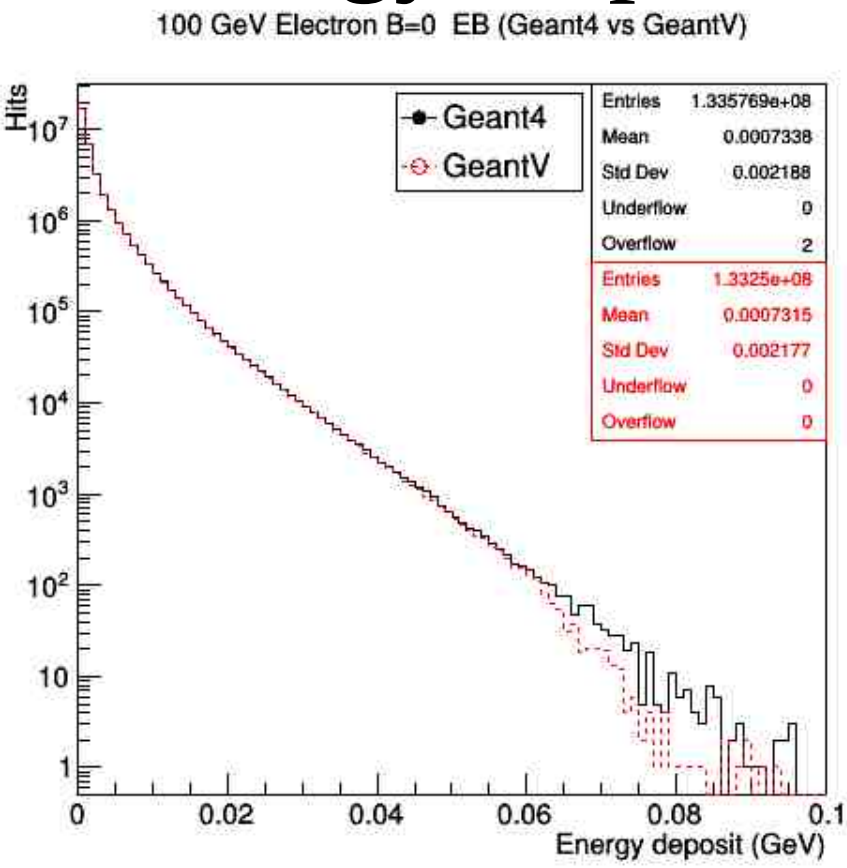
Physics Validation

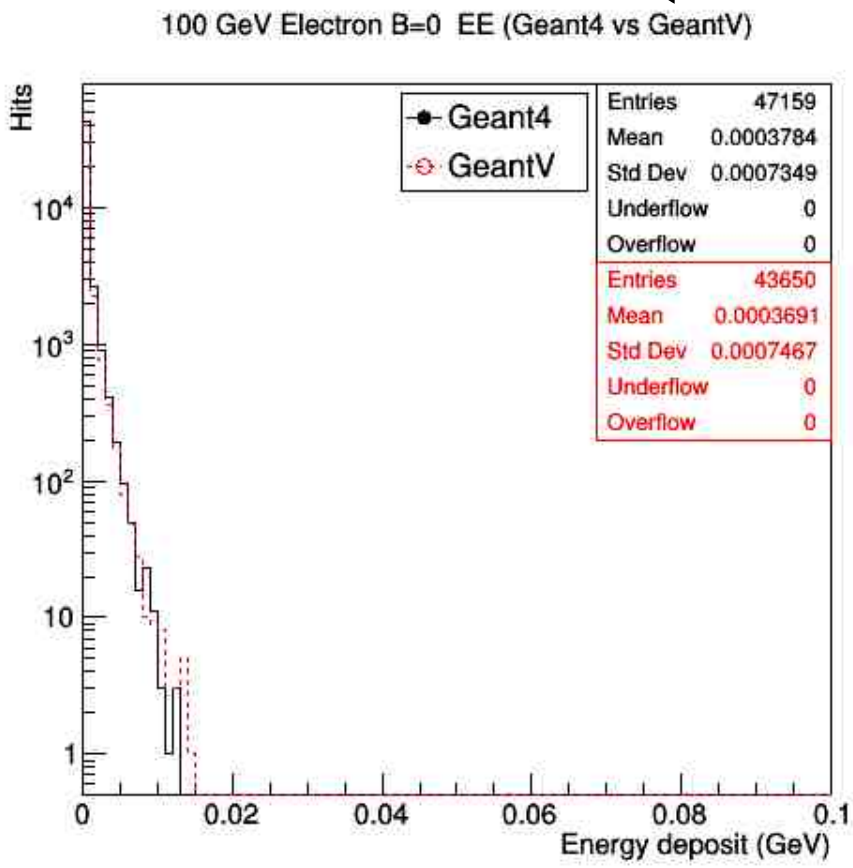
- Settings:
  - Geant4 10.4p2 w/ VecGeom v0.5 (scalar)
  - GeantV pre-beta-7 w/ VecGeom v1.1
  - All CMS-specific G4 optimizations disabled
  - Same production cuts (default 1mm)
  - Single thread (reproducible pRNG sequences)
  - Generate 1000 events w/ single electron, E = 100 GeV, $\eta = 1.0$, $\varphi = 1.1$
- Tests: (same generated events used for G4 and GV)

1. No field (B = 0)
2. Constant field (B = 3.8 T)

(more in backup)

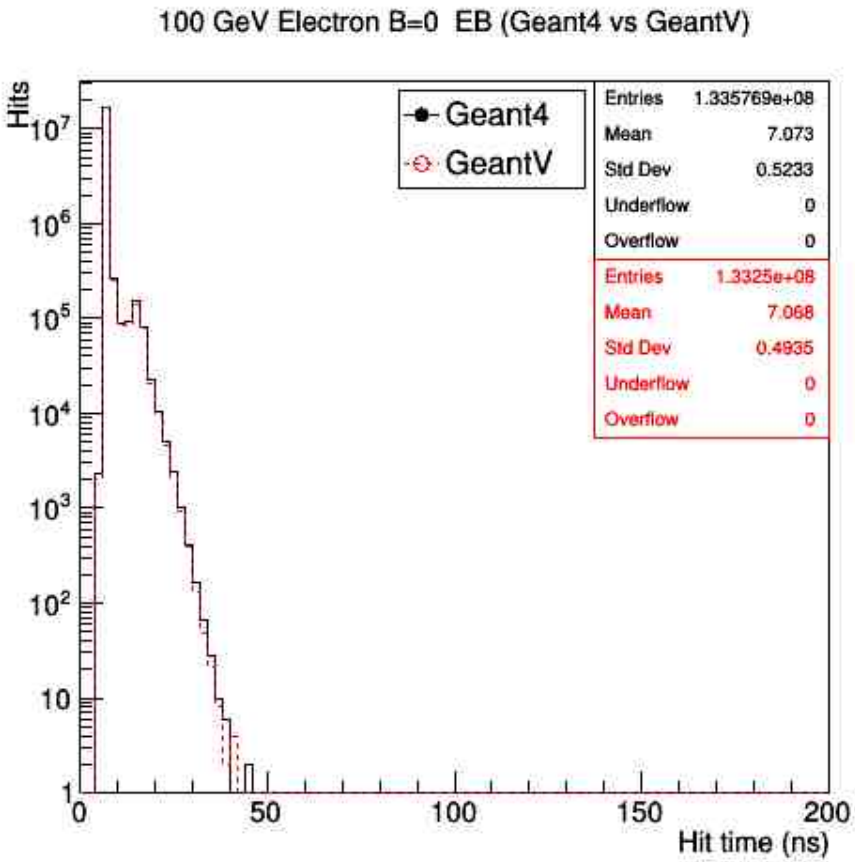
1. Energy Deposits for 100 GeV e- (B=0)

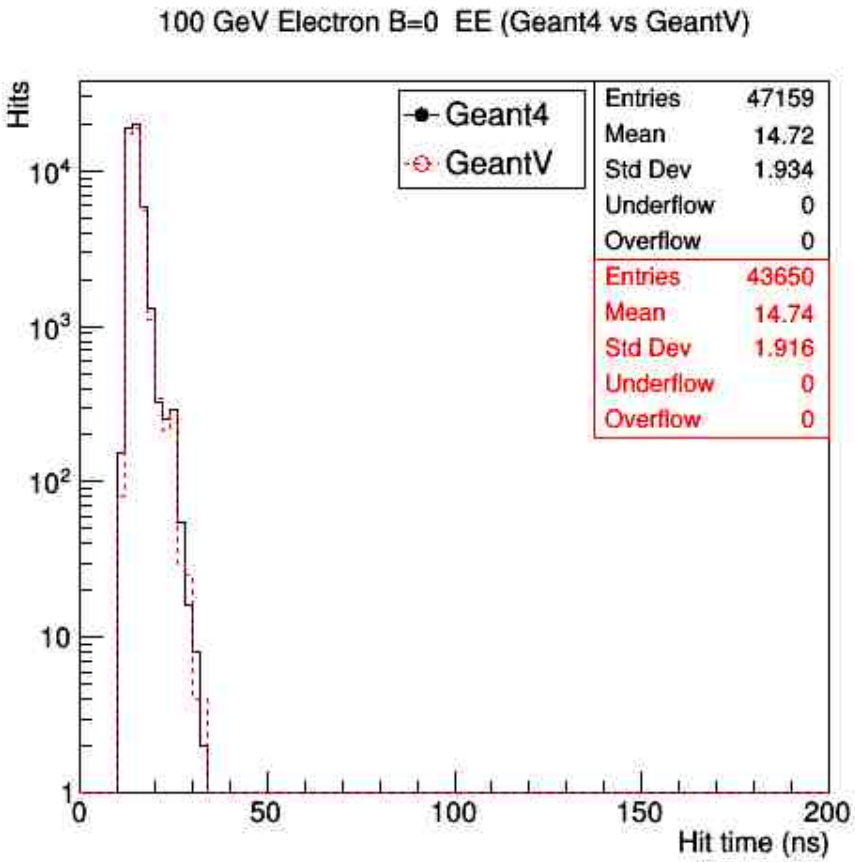




- The number of entries differs by 0.3% (7.4%) in EB (EE)
- The means differ by 0.2% for EB and 2.5% for EE

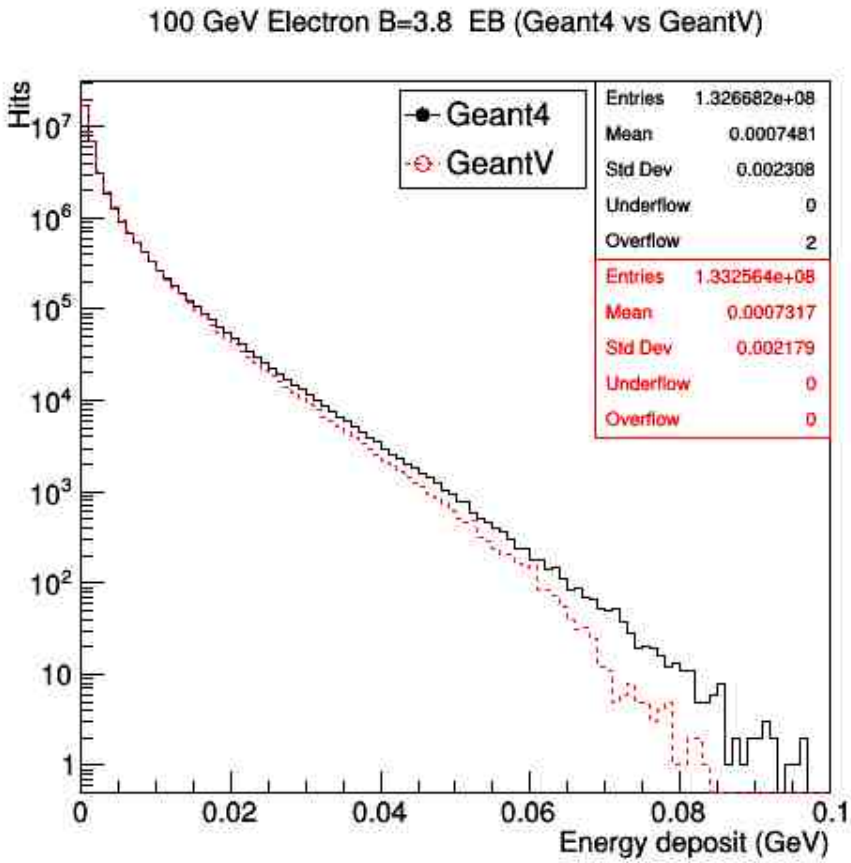
1. Hit Time for 100 GeV e- (B=0)

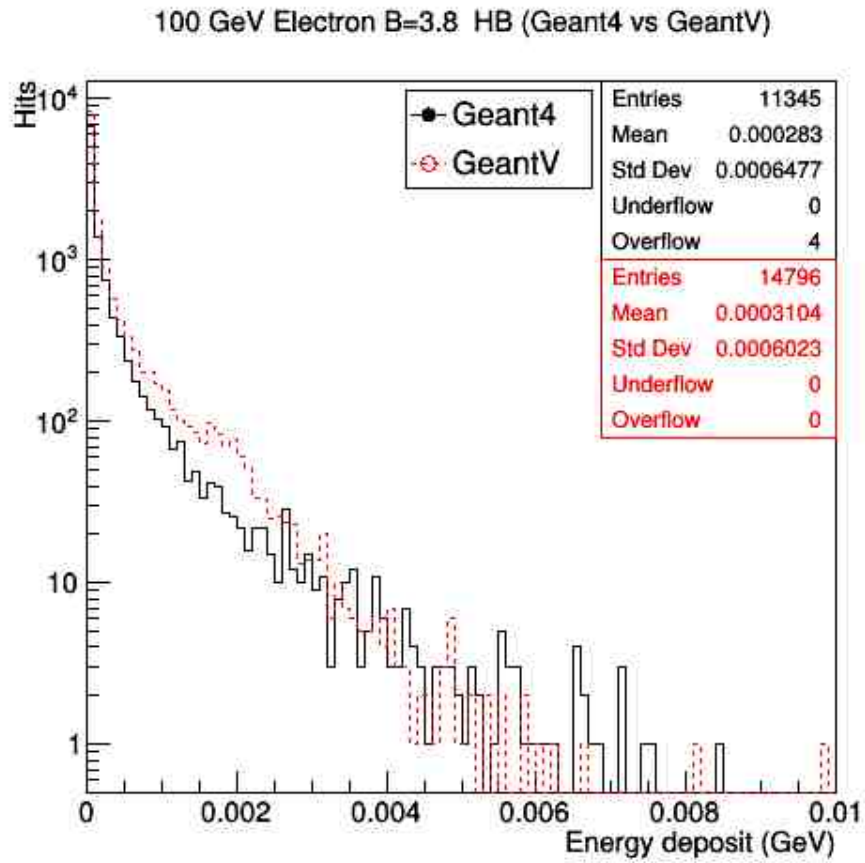




- Means differ by 0.07% for EB and 0.13% for EE
- GeantV and Geant4 applications provide roughly the same distributions

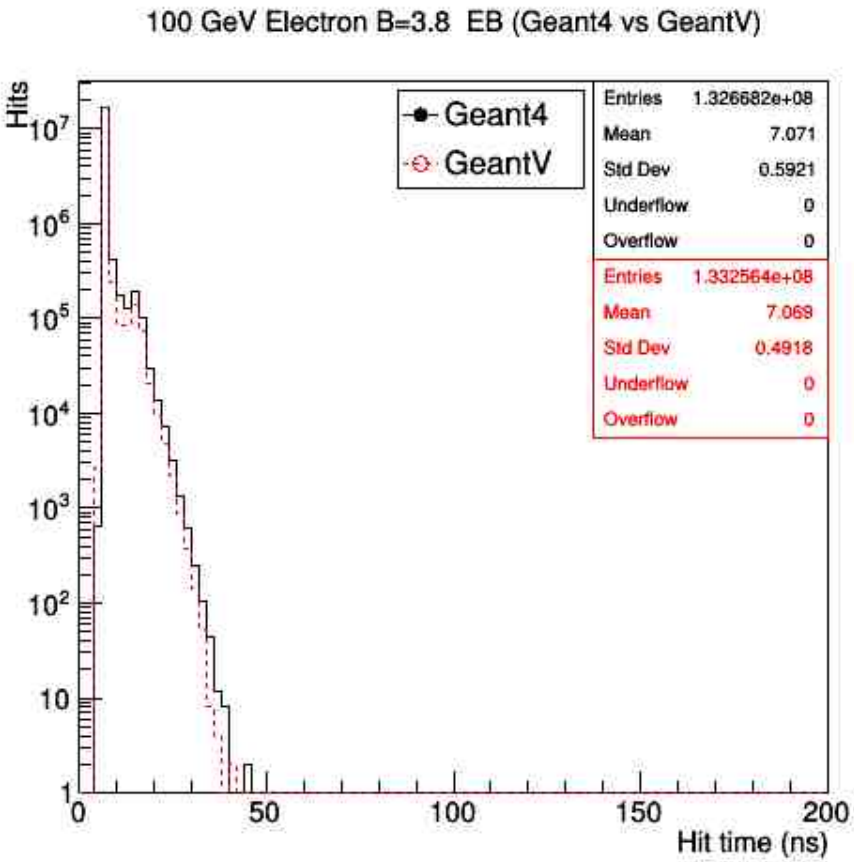
2. Energy Deposits for 100 GeV e- (B=3.8)

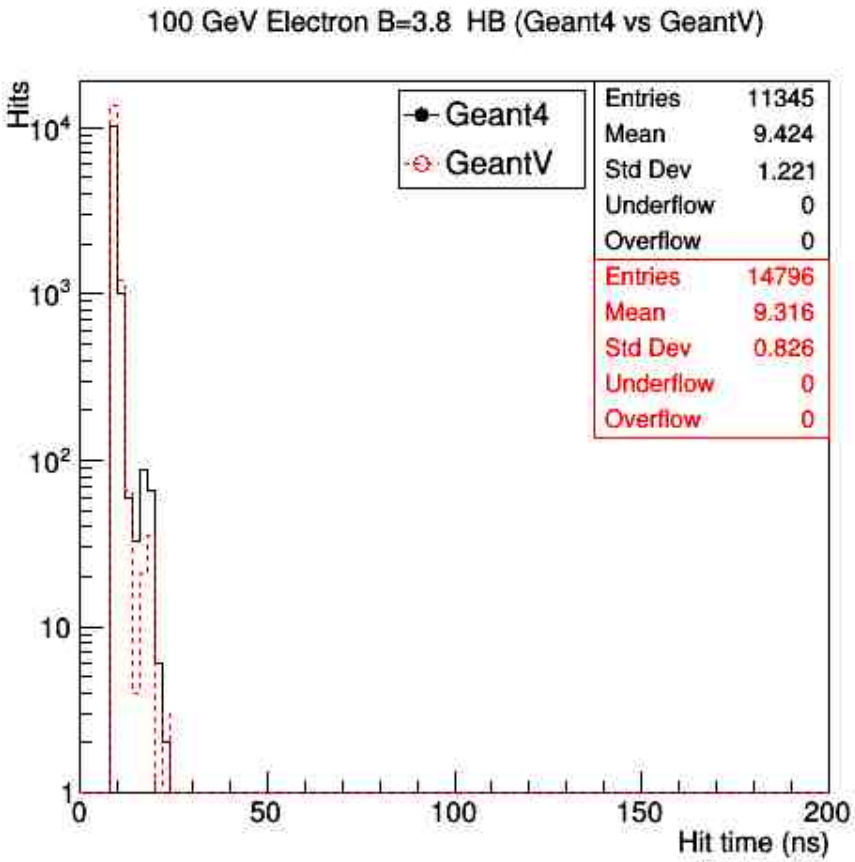




- The number of entries differ by 0.4% (23.3%) in EB (HB)
- The means differ by 2.2% for EB and 8.8% for HB

2. Hit Time for 100 GeV e- (B=3.8)





- The means differ by 0.03% for EB and 1.15% for HB
- There is a small difference in the physics results of GeantV and Geant4 applications in the presence of B-field

Performance Tests

- Settings:
  - GeantV pre-beta-7+ (63468c9b)
    - Enabled: vectorized multiple scattering, field (not physics)
  - Generate 500 events, 2 electrons w/ E = 50 GeV, random directions
  - Keep # events / thread constant (copy & concat 500 generated events)
  - Use same generated events in G4 and GV
  - Keep unused threads busy
  - Disable output
- Machine: FermiCloud VM w/
  - Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz, 4096 KB cache
  - sse4.2 instructions
- Track wall clock time & memory with CMSSW TimeMemoryInfo tool
  - Measures VSIZE, RSS per event
  - Calculate speedup from wall time
    (divided by # threads used, since # events / thread is constant)

Characterization

- VM CPU has relatively small cache
  - Known that major component of GeantV speedup arises from smaller library → fewer cache misses

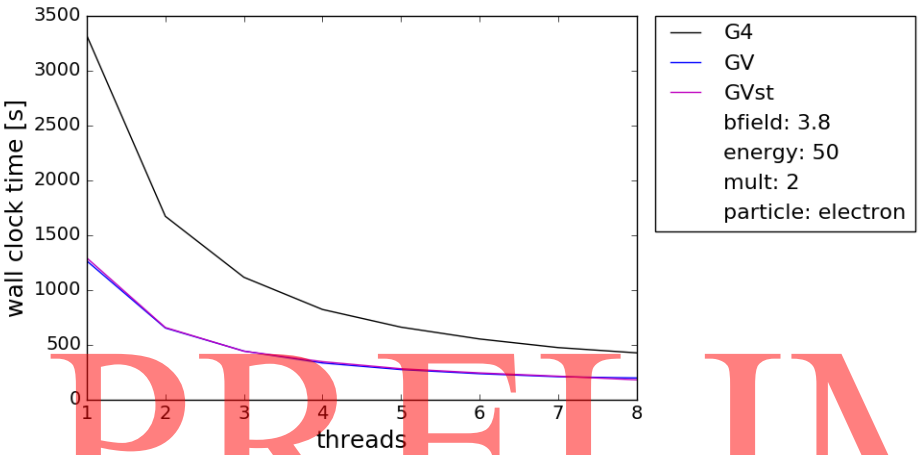
➢ To characterize CMSSW performance results, first run built-in GeantV FullCMS standalone test

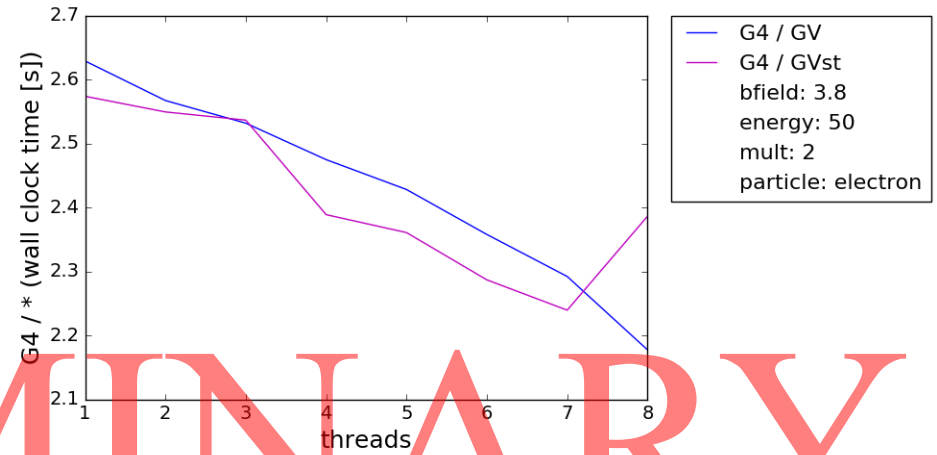
  - Single thread, settings as close to previous slide as possible (see test script: testStandalone.sh)
  - NB: different physics list used in standalone vs. CMSSW

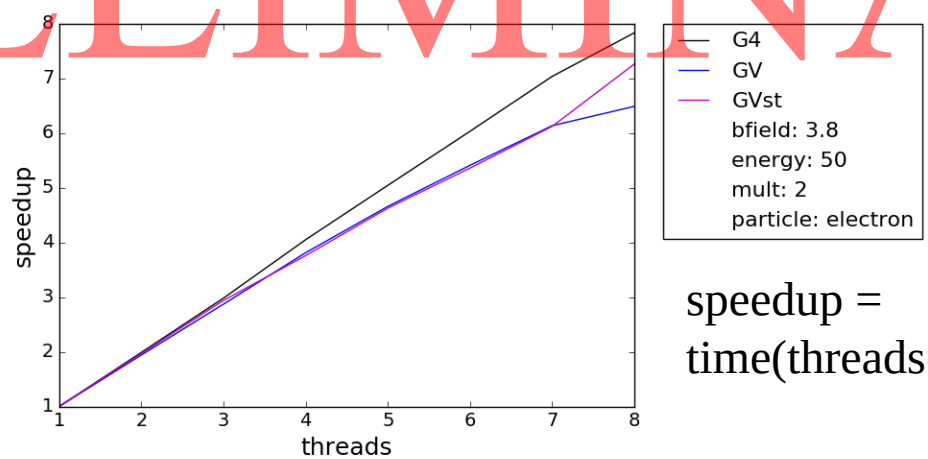
- Results:
  - GeantV: RealTime=756.002s CpuTime=753.09s
  - Geant4: User=1617.36s Real=1618.52s

→ 2.14× speedup (standalone)

Time Performance



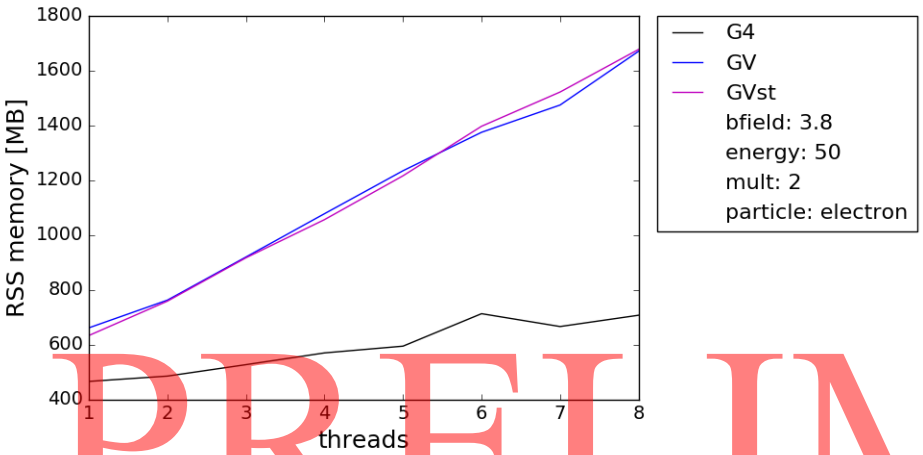


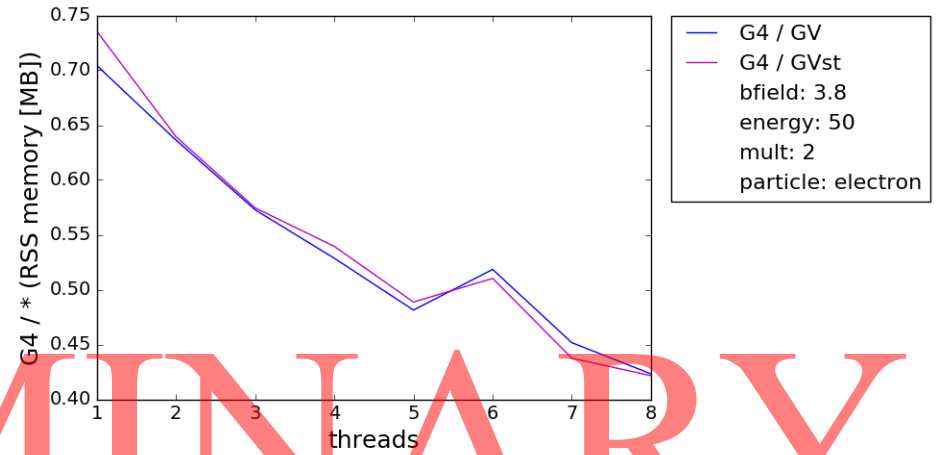


speedup = time(threads=1)/time(threads=N)

- GV 2.6× faster than G4 single thread, still ~2.2× faster in MT
- GV single track mode similar to basketized
- G4 has better scaling w/ # threads than GV

Memory Performance





- Memory grows ~linearly w/ # threads (expected)
- GV uses more memory than G4 (expected)
- Single track mode uses similar memory to basketized

What Would Be Next?

- To complete the goals of CMS R&D studies for the paper:
  - Full magnetic field map
  - Test on machines w/ different cache sizes
- Stretch goals/notes for future similar projects:
  - Random number generator
  - Adapt scoring classes for other detectors (beyond calorimeters)
  - Combine w/ other simulation improvements
    - Notably Russian Roulette & HF shower library, which give largest gains
- If GeantV project were to continue:
  - Better solution for geometry conversion than TGeo
  - Sensitive volume/detector functionality
  - Vectorized hadronic physics
  - Improve threading, memory management, and ownership models
  - Decouple event loading & task launching in ExternalLoop mode
  - Event-wise scoring rather than current thread-wise scoring w/ TaskData

Conclusions

- CMS studies met ~all goals laid out
  - Co-development led to improvements and bug fixes in GeantV to facilitate experiments' use
  - One of the first projects to exercise CMSSW ExternalWork feature
  - Physics validation & CPU measurements show very positive results
  - Path to adapt interfaces efficiently is laid out:
    "Rosetta stone" mostly contained in StepWrapper and VolumeWrapper

Geant4	GeantV
StepWrapper	StepWrapper
VolumeWrapper	VolumeWrapper

- Demonstrator to test major elements of GeantV-CMSSW integration is ready
  - Up to 2.6× speedup in CMSSW application
  - Will finalize results for paper

Backup

Template Wrappers

- **Goal**: use *exact same* SD code for Geant4 and GeantV
- Problem: totally incompatible APIs
  - Example: `G4Step::GetTotalEnergyDeposit()` vs. `geant::Track::Edep()`
- Solution: template wrapper with unified interface
  e.g. `StepWrapper<T>::getEnergyDeposit()`
  - SD code *only calls* the wrapper
  - Wrapper stores pointer to T (minimize overhead)
- Current wrappers:
  - BeginRun
  - BeginEvent
  - Step
  - Volume
  - EndEvent
  - EndRun

Traits

- Collect Geant4/GeantV-specific types and wrappers into unified **Traits** class:

```
struct G4Traits {
  typedef G4Step Step;
  typedef sim::StepWrapper<Step> StepWrapper;
};
struct GVTraits {
  typedef geant::Track Step;
  typedef sim::StepWrapper<Step> StepWrapper;
};
```

- Provides standardized typenames to be used by SD class:

```
template <class Traits> class CaloSteppingActionT : …,
  public Observer<const typename Traits::Step *>
{
  public:
    void update(const Step * step) override {
update(StepWrapper(step)); }
  private:
    // subordinate functions with unified interfaces
    void update(const StepWrapper& step);
};
```

Organization

Old

CaloG4
CaloSteppingAction (.h, .cc)

New

Calo
CaloSteppingActionT (.h, .icc)
Wrappers (.h)

CaloG4
CaloSteppingAction (.h, .cc)
G4 Wrappers (.h), Traits (.h)

CaloGV
CaloSteppingAction (.h, .icc)
GV Wrappers (.h), Traits (.h)

- SD interface & implementation in **Calo** (.icc file), w/ unimplemented wrapper interfaces
- G4/GV wrapper specializations in **CaloG4/GV**, w/ specific instances of templated SD class → isolate dependencies

Scoring Approaches

- Two approaches to scoring in CMSSW:

1. Inherit from **G4VSensitiveDetector** (Geant4 class)
   → automatically initialized for geometry volumes marked as sensitive
2. Inherit from **SimWatcher** (CMSSW standalone class)
   → need to specify names of watched geometry volumes

- CaloSteppingAction is a demonstrator class w/ approach 2
  - Simplified version of ECAL and HCAL scoring
  - Less dependent on Geant4 interfaces
- "Real" SD code uses approach 1

➢ More work to extract Geant4 dependencies will be necessary

- Some SD class methods directly from Geant4 (via inheritance)
- Need to mock up Geant4-esque interfaces w/ dummy classes for GeantV

More Physics Validation

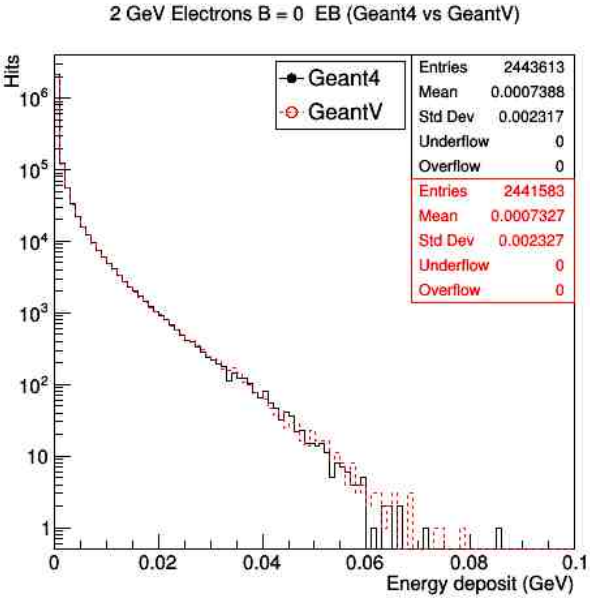
3. Generate 1000 events of single electrons at 2, 10 and 50 GeV at a fixed direction and compare GeantV against Geant4 with magnetic field off and on at 3.8 Tesla

4. Generate 100 events of 50 GeV double electrons at 50 GeV with $-3 < \eta < 3$ and $0 < \varphi < 2\pi$, run in multi-threaded mode (4 threads), B = 0 Tesla

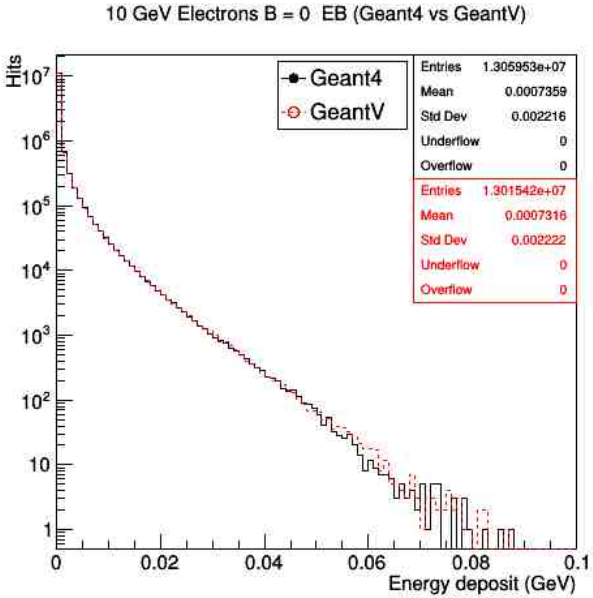
5. Repeat multi-threaded test with B = 3.8 Tesla

3. Energy Deposit with B = 0

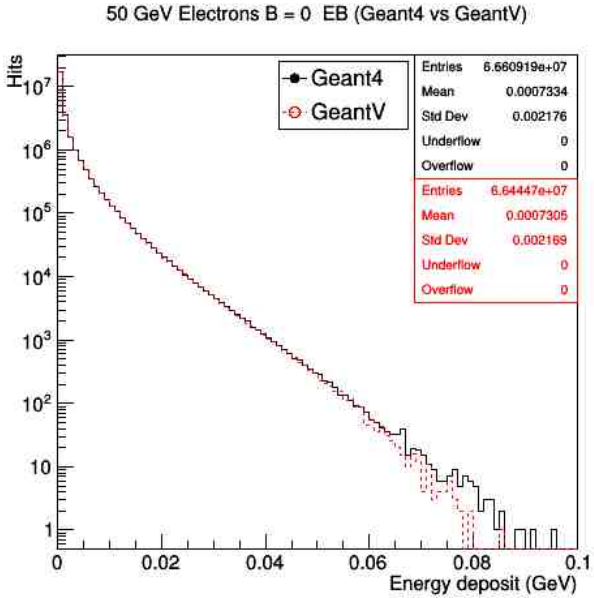
2 GeV Electrons



10 GeV Electrons



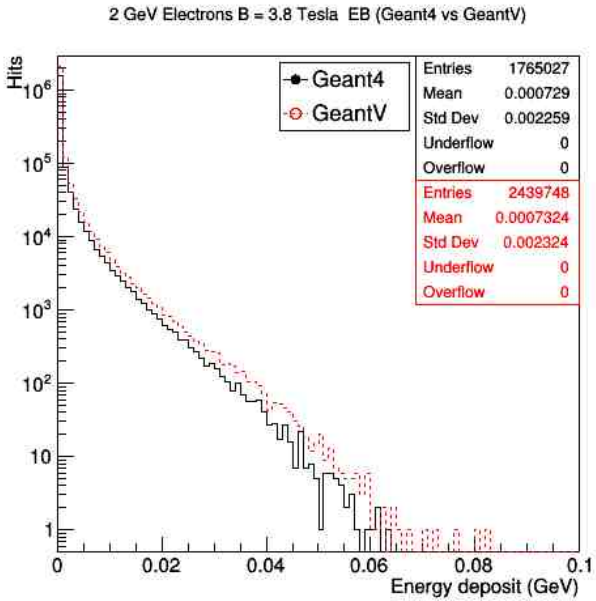
50 GeV Electrons



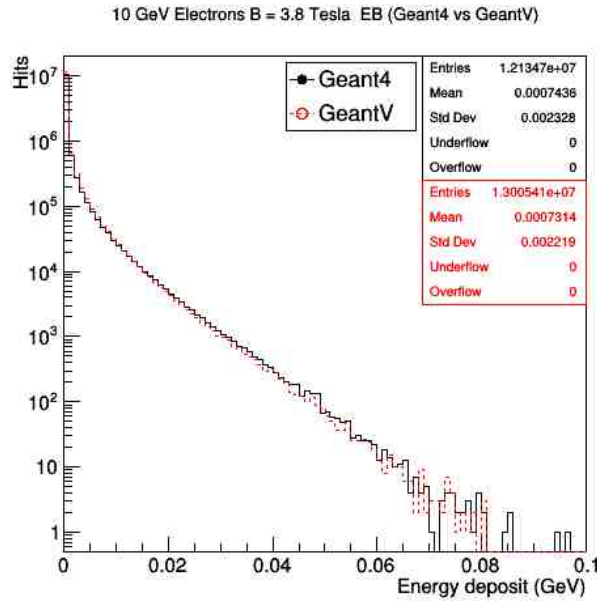
- Number of hits is the same for all 3 energies. The differences are at the level of 0.1/0.3/0.2% for 2, 10 and 50 GeV
- The means differ by 0.8/0.6/0.4% at the three energies

3. Energy Deposit with B = 3.8

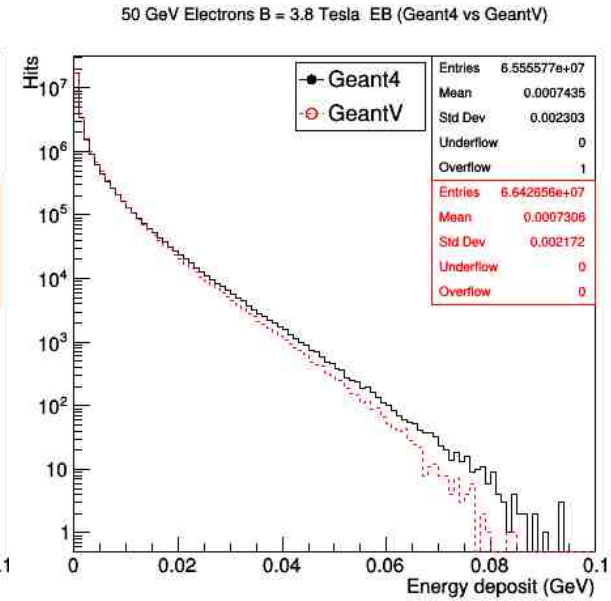
2 GeV Electrons



10 GeV Electrons

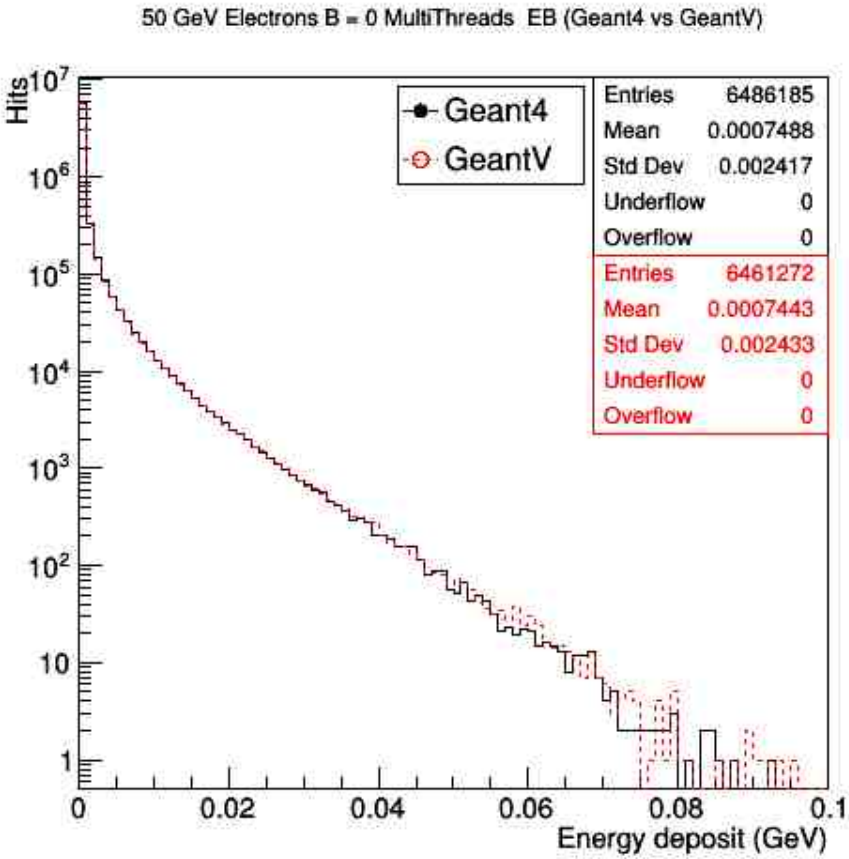


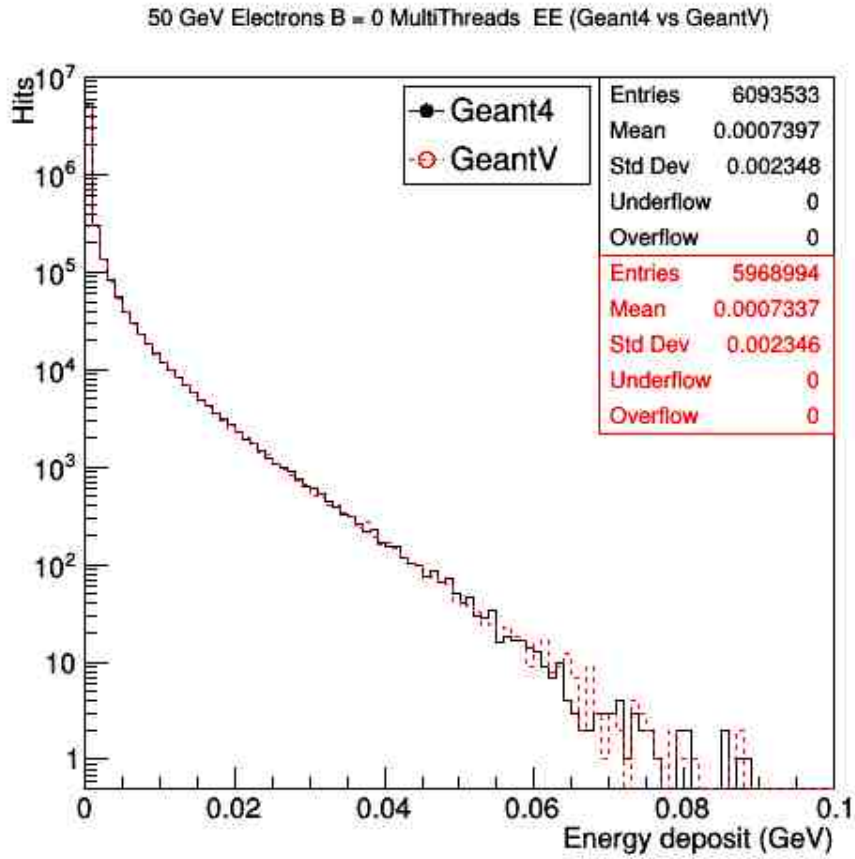
50 GeV Electrons



- Number of hits is the same for all 3 energies. The differences are at the level of 27.7/6.7/1.3% for 2, 10 and 50 GeV
- The means differ by 0.5/1.6/1.7% at the three energies

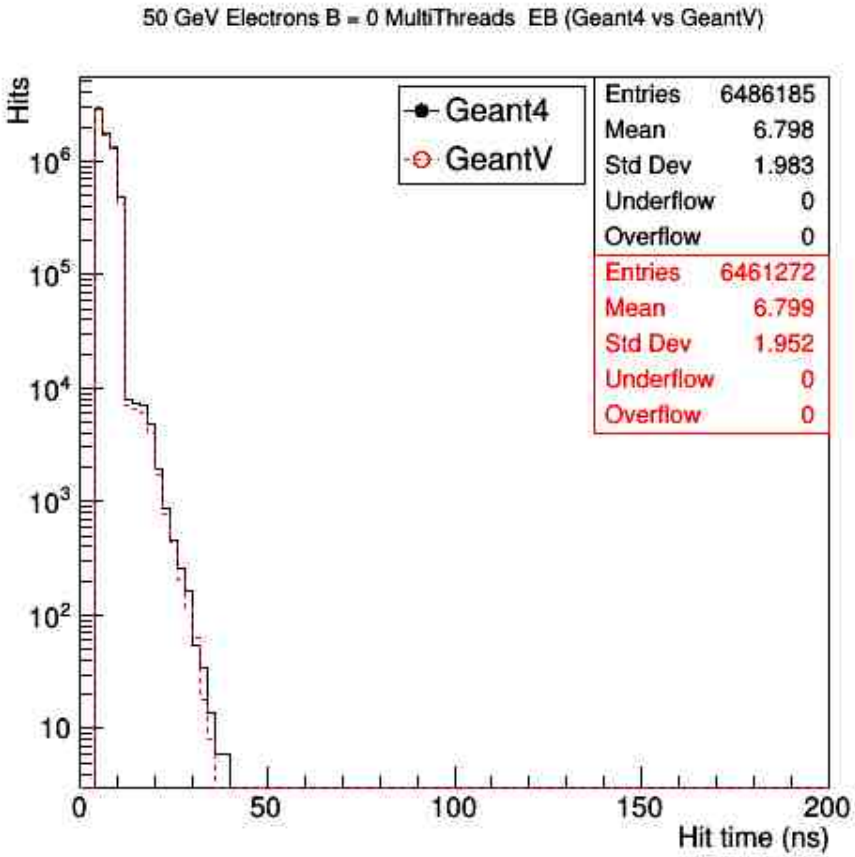
4. Energy Deposit with B = 0, MT

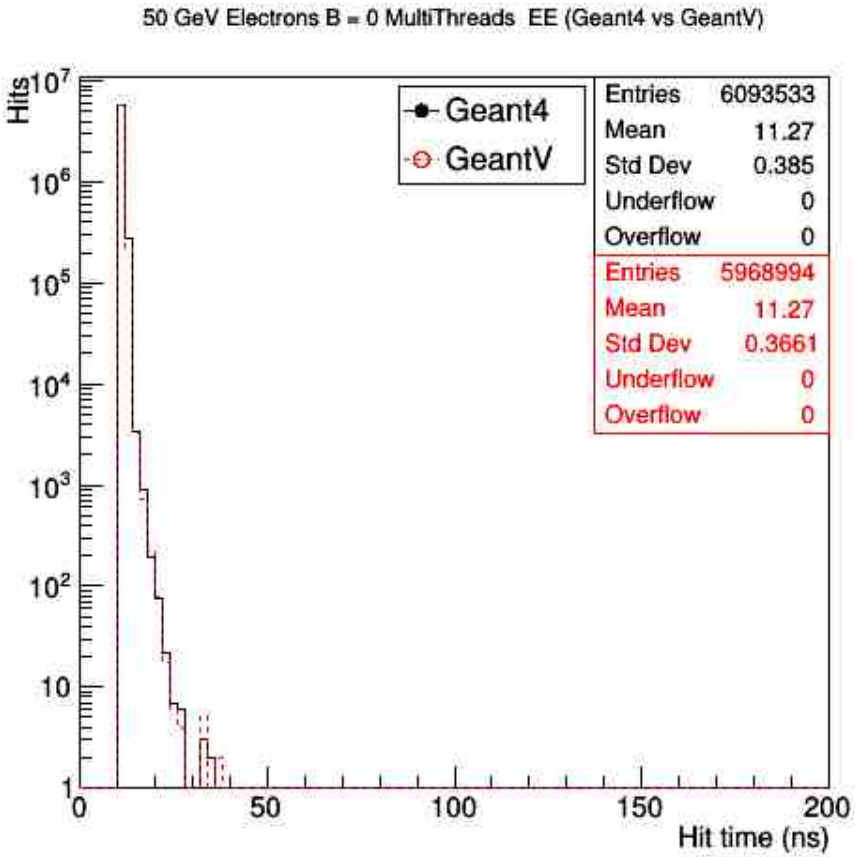




- Events are generated with 50 GeV electrons having random direction within a limited range of η and φ
- The agreement is pretty good in the B=0 option for both # of hits as well as in the shape of the distributions for EB and EE

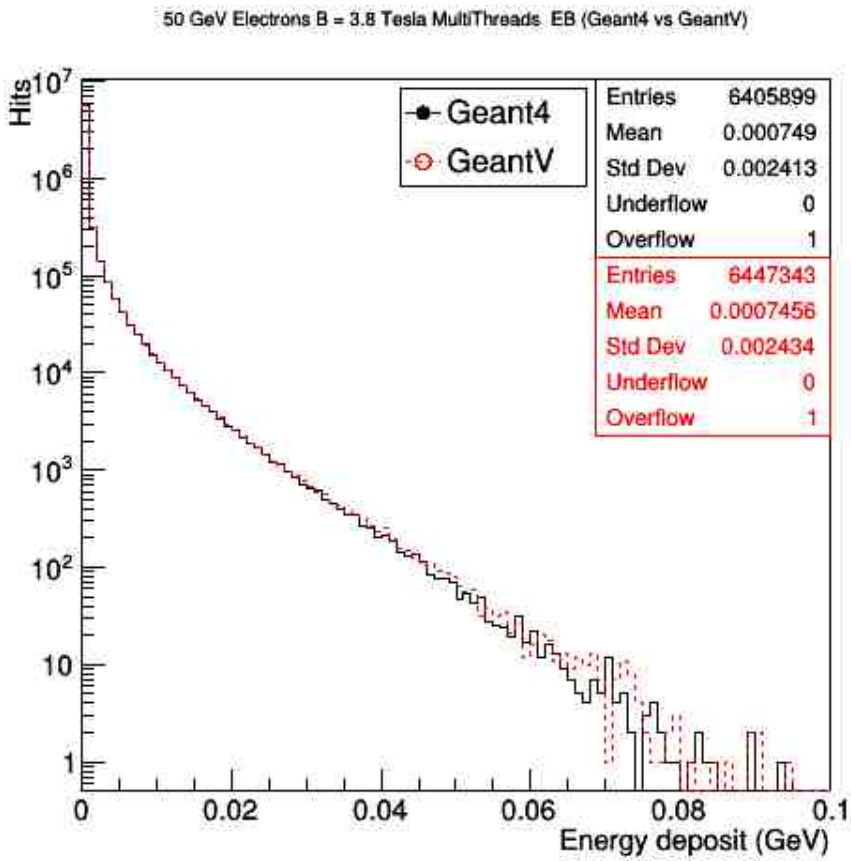
4. Hit Times with B = 0, MT

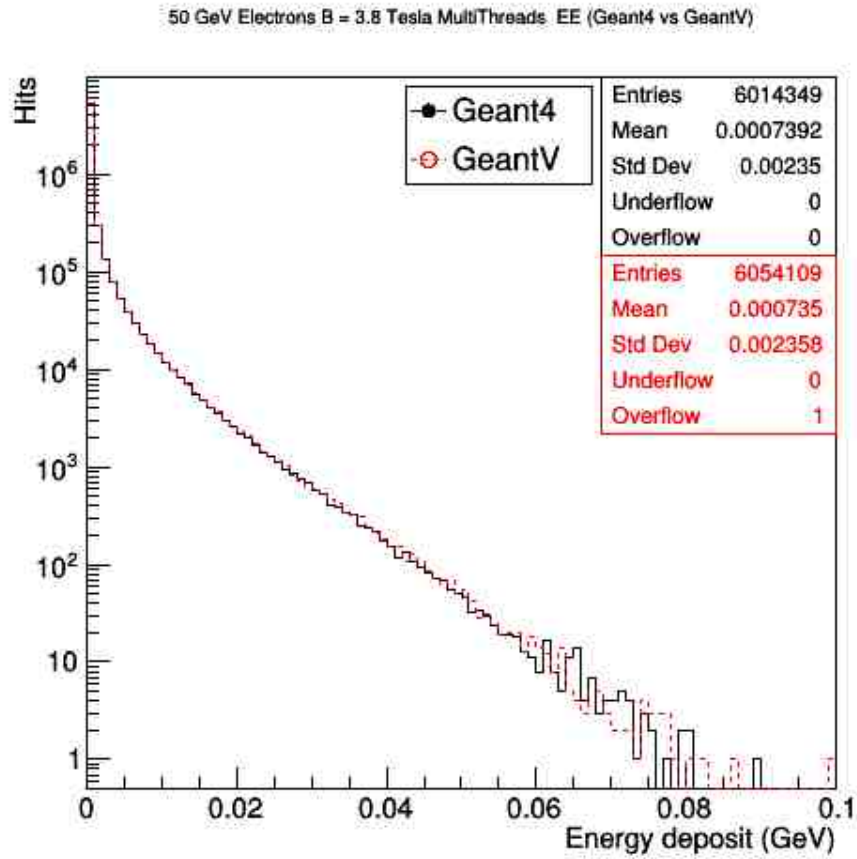




- Hit time distributions are also in good agreement for the B=0 option in EB as well as in EE

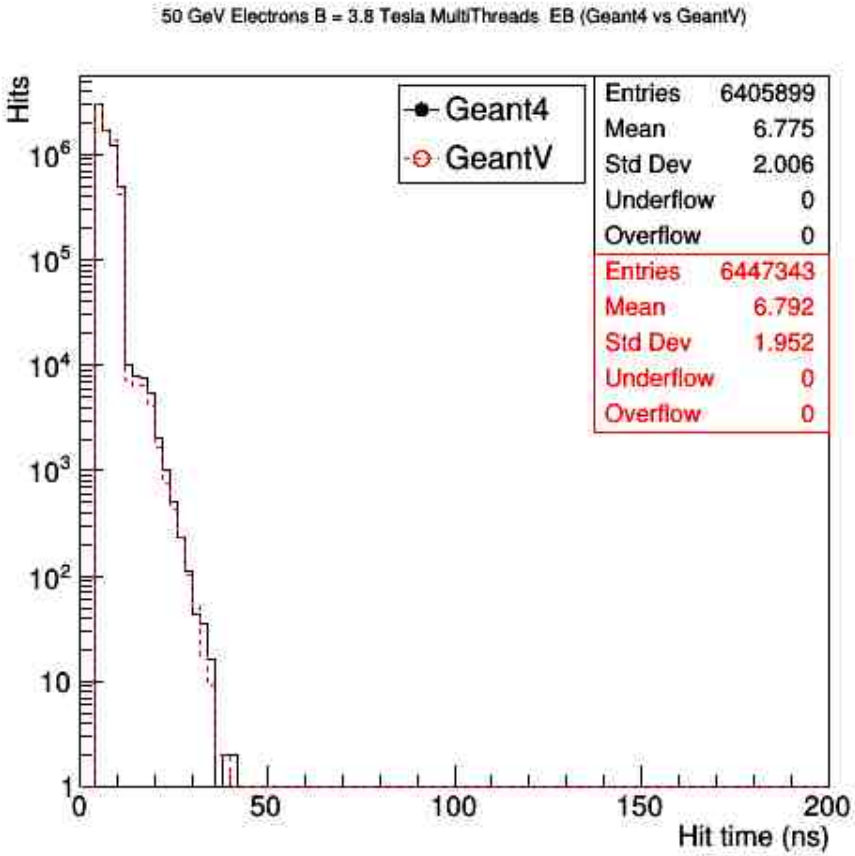
5. Energy Deposit with B = 3.8, MT

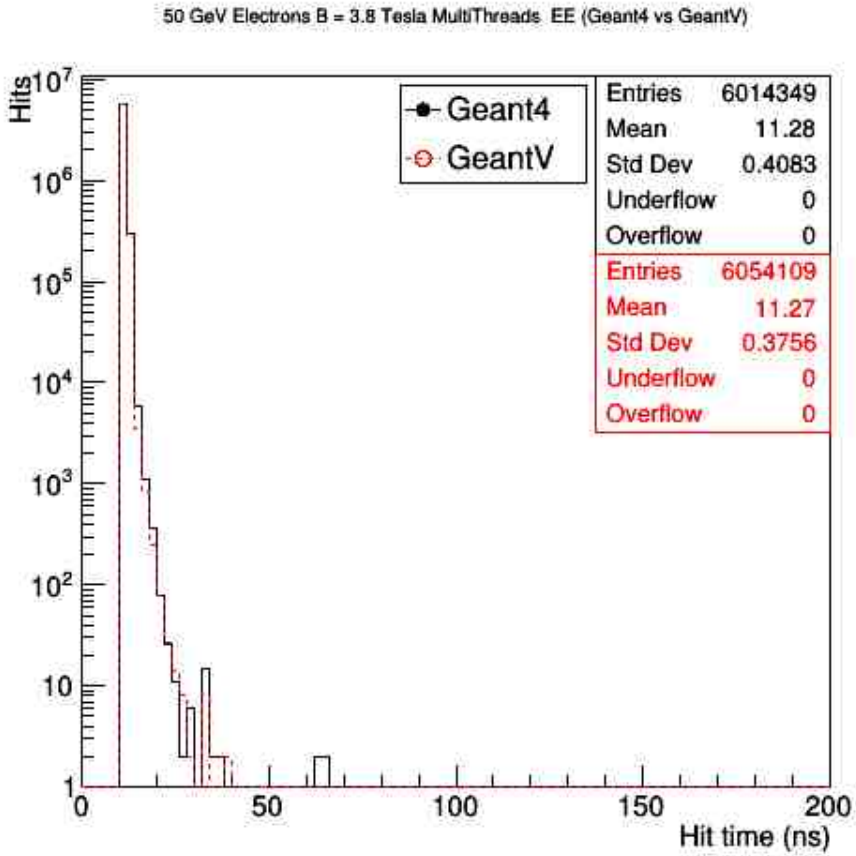




- Same events (50 GeV electrons, random direction within a limited range of η and φ) are simulated in a uniform B-field option of 3.8 Tesla
- The agreement is still good for both # of hits as well as in the shape of the distributions for EB and EE

5. Hit Times with B = 3.8, MT





- Hit time distributions are also in reasonable agreement for the B = 3.8 Tesla option in EB as well as in EE

CMS Simulation Optimizations

	Relative CPU usage	
Configuration	**MinBias**	**ttbar**
No optimizations	1.00	1.00
Static library	0.95	0.93
Production cuts	0.93	0.97
Tracking cut	0.69	0.88
Time cut	0.95	0.97
Shower library	0.60	0.74
Russian roulette	0.75	0.71
FTFP_BERT_EMM	0.87	0.83
VecGeom (scalar)	0.87	0.93
Mag. field step,track	0.92	0.90
All optimizations	0.16	0.24

- HF shower library, Russian Roulette have largest impacts
- VecGeom, mag. field improvements entered production in past ~year
  - Enabled by validating and using latest Geant4 versions
- Cumulative effects: overall, simulation is **6.2×** (**4.1×**) faster for MinBias (ttbar) vs. default Geant4 settings

➢ CMS full simulation is at least 8× faster than ATLAS