John Apostolakis/CERN, Andrea Dotti/SLAC, Krzysztof Genser/FNAL, Soon Yung Jun/FNAL, Boyana Norris/Univ. of Oregon and ANL

US HEP-ASCR Fermilab Meeting; February 2014

Preliminary results of Geant4 Electromagnetic Code Review

presented by K. L. Genser

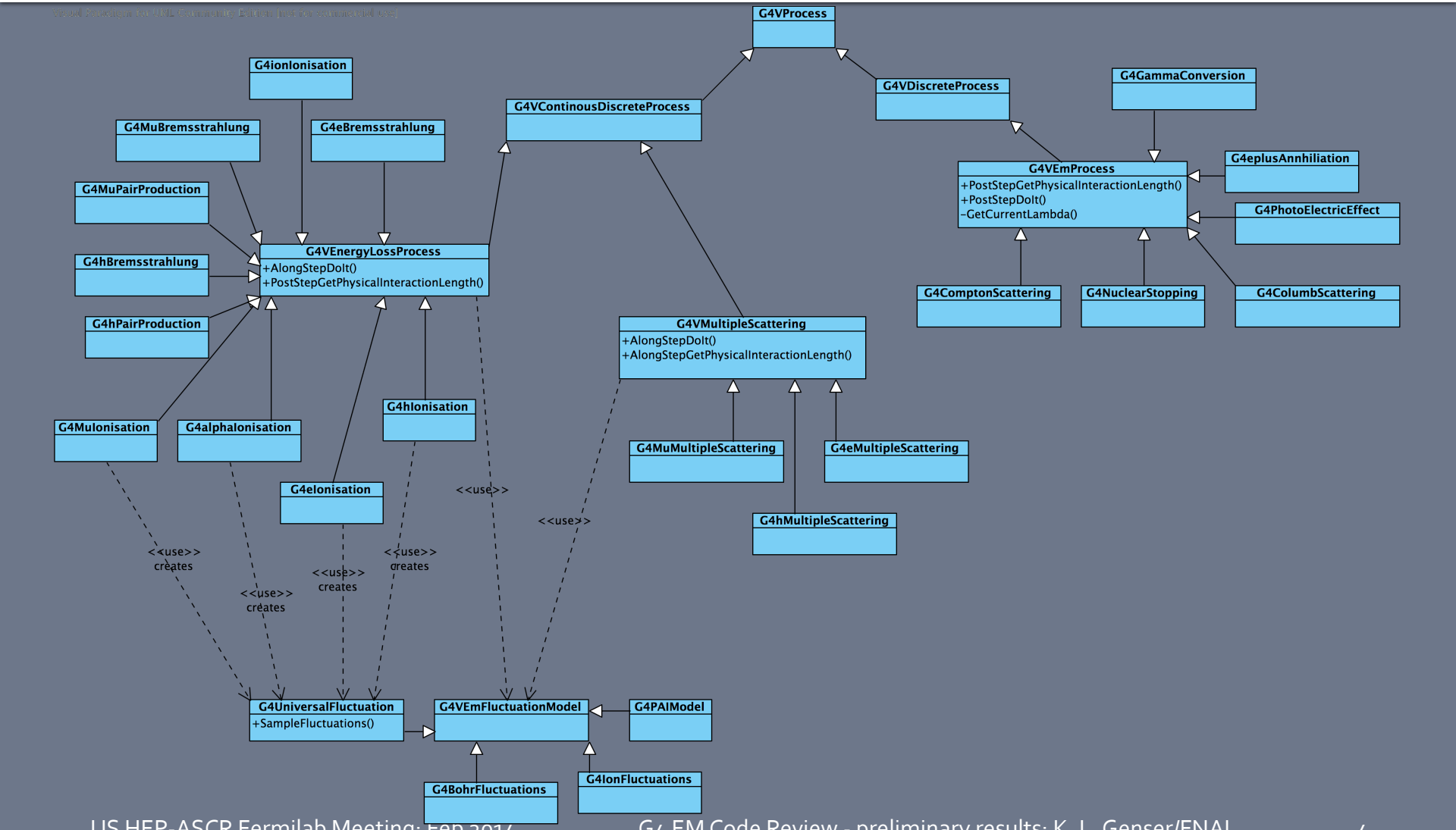
Outline

- Initial Scope of Review
- EM Related Classes
- Tested Applications and Test Tools
- Results for selected Classes/Functions
- Other Observations/Suggestions
- Plans
- Summary

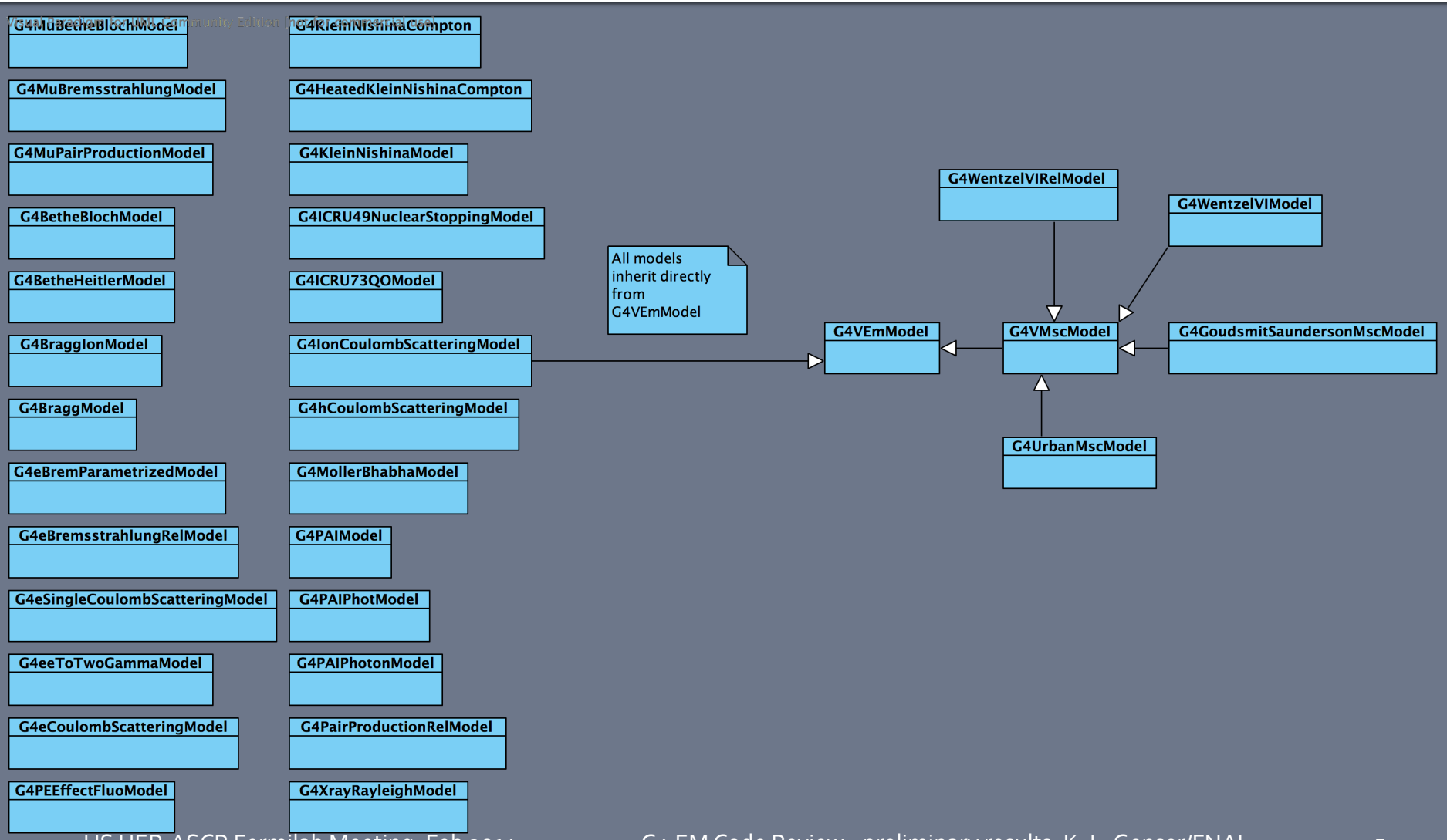
Scope, Initial Plans, Timeline

- Plan as formulated in Spring of 2013
  - Review of performance aspects of a subset of ElectroMagnetic (EM) and closely related classes of Geant4 code with the initial goal to assess if the code is written in a computationally optimal way and to see if it could be improved, keeping in mind
    - correctness, performance, maintainability and adaptability
    - Multi-Threading aspects
    - potential issues related to parallelization and/or migration to GPUs
    - issues or potential improvements related to future migration to C++11
  - The review should initially concentrate on the most CPU costly classes and functions
  - After the initial phase it may be needed or useful to expand the scope of the review to other related areas or aspects of the code
- There was an intermediate report delivered to the Geant4 Collaboration at the yearly meeting in September 2013

G4VProcess Class Diagram



G4VEmModel Class Diagram



Initial List of Functions to be reviewed

- G4PhysicsVector, esp. (Compute)Value
  G4PhysicsLogVector, esp. FindBinLocation
- G4VProcess, esp. SubtractNumberOfInteractionLengthLeft
- G4VEmProcess, esp. PostStepGetPhysicalInteractionLength, GetCurrentLambda, PostStepDoIt
- G4VEnergyLossProcess esp. PostStepGetPhysicalInteractionLength, AlongStepDoIt, GetLambdaForScaledEnergy, AlongStepGetPhysicalInteractionLength
- G4VMultipleScattering esp. AlongStepDoIt, AlongStepGetPhysicalInteractionLength
- G4VEmModel, G4VMscModel, G4UrbanMscModel(95) esp. ComputeGeomPathLength, ComputeTruePathLengthLimit, SampleCosineTheta, SampleScattering

Geant4 Versions/Tested Application

- Geant4 v9.6.r07 was the basis of the initial work; working with v10 now
- Settled on using SimplifiedCalo as the executable which performance was analyzed to study effects of code transformations
  - allowed to concentrate on the EM code and to minimize the importance other factors
    - if not doing full Geant4 profiling most of the test were done with FTFP_BERT physics list ;50GeV e-; no magnetic field
  - we have made sure that, while testing our changes, the final random number stayed the same after we modified the original code

Test Cluster and Tools

- 5 node x 4x8 Core AMD Opteron Processor 6128 (CPU 2000 MHz) cluster
  - L1 Cache Size 128KB
  - L2 Cache Size 512KB
  - L3 Cache Size 12288KB
  - 64GB memory on each node
- gcc v4.4..8
- fast (sampling) profiler for standard profiling/benchmarking
  - https://cdcvs.fnal.gov/redmine/projects/fast
  - https://oink.fnal.gov/perfanalysis/g4p/admin/task.html
  - Added new table to the standard profiling data to be used as an input to TAU and similar tools
- TAU (Tuning and Analysis Utilities) Performance System; also used HPCToolkit and Open|SpeedShop

Results for Selected Classes/Functions

G4PhysicsVector (in v9.6.r07)

- G4PhysicsVector ("container class" used heavily e.g. via G4PhysicsTable)
  - in order to collocate the data which is used together, replaced three main data members ("data", "bin", "secDerivative") of std::vector<G4double> type) with one std::vector<G4xyd>
    - G4xyd – a helper class with three G4double data members, operator<, and default () and 3 argument constructors(c'tors), (G4double x,G4double y,G4double d)
    - i.e. replaced three vectors with one vector of structs to localize access
  - initialized all data members in the constructors (c'tors), removed copy c'tor and (assignment) operator= as the default ones supplied by the compiler should be correct eliminating the need to maintain the hand written code
  - modified derived classes accordingly
  - replaced some ifs with the ?: (ternary) operator
  - replaced hand-coded binary search in G4LPhysicsFreeVector::FindBinLocation with std::lower_bound

G4PhysicsVector (v9.6.r07 vs. v10)

- The overall effect of transforming G4PhysicsVector in v9.6.r07 was about 1.5% performance improvement as measured using standard profiling/benchmarking tools used to profile production/ reference Geant4 releases;
  - the G4PhysicsVector::Value function itself used to take about 3% of the total execution time (pythia higgs events)
  - the timing improvement did not occur in that function itself but in other areas, mainly in CLHEP::MTwistEngine::flat likely due to cache effects
- No CPU effect in Geant4 v10 for the equivalent transformation
  - in G4 v10, FindBinLocation was moved to the base class and inlined by the author (an if was used to detect the type (using an enum) of the underlying classes)
    - after inlining FindBinLocation G4PhysicsVector::Value function itself takes about 6% of the total execution time now (pythia higgs events)
    - noted a potential typo in one of the type enums (looks confusing, but seems harmless)

G4Physics2DVector

- G4Physics2DVector
  - investigated changing the type of the underlying container to float
    - in unit tests it gave ~13% degradation on CPU, ~12% improvement on an NVIDIA GPU
  - changed the underlying container std::vector<std::vector<G4double>*> to

    std::vector<std::vector<G4double> > which allowed to remove copy c'tor, operator= and destructor
  - used std::lower_bound in FindBinLocation and inlined the function
  - based on the ~13% improvement in a unit test, expected a minimal overall performance improvement, but the result was more than ~5% degradation despite G4Physics2DVector::Value function taking only ~0.3% of the execution time, again likely due to cache effects; seems to be confirmed by the cache miss data; no partial specialization in stl
  - Data layout may be more important than the algorithms
    - Profiling/benchmarking after each change is important
      - stack analysis alone may not be sufficient to study cache effects

G4UniversalFluctuations:: SampleFluctuations

- G4UniversalFluctuations::SampleFluctuations
  - Tested using random number generator array API:

replaced

```
for (G4int k=0; k<nb; k++) lossc += w2/(1.-w*G4UniformRand());
```

(G4UniformRand() is a macro:
G4Random::getTheEngine()->flat(); returning one random number)
with

```
G4double rannums[nb];
G4Random::getTheEngine()->flatArray(nb,rannums);
for (G4int k=0; k<nb; k++) lossc += w2/(1.-w*rannums[k]);
```

  - About 10% faster in unit test at the SampleFluctuations level when the pointer to the engine was also cached

Other changes related to SampleFluctuations, e.g. decreasing number of divisions, factorizing constants, result in about ~3% improvement in the unit test; Noted a potential small bug in the algorithm; author notified

G4UrbanMscModel & G4Poisson

- G4UrbanMscModel
  - Inlined shorter functions of (about 1.5% overall effect; comparable to the original time spent in those functions)
    - SampleDisplacement()
    - LatCorrelation()
    - ComputeThetao(double, double)
    - SimpleScattering(double, double)
  - noted "parallel" usage of G4Exp & std::exp functions
- Inlined G4Poisson free function
  - gained 30+% CPU in a unit test and about 2% overall

G4VEnergyLossProcess derived classes

- In G4VEnergyLossProcess which is a G4VContinuesDiscreteProcess all derived classes except G4eIonization and G4IonIonization should probably be derived from G4VDiscreteProcess
  - i.e. AlongStepGetPh…Int…Lenght and AlongStepDoIt are not necessary.
    - The isIonisation flag is set to true in the constructor of G4VEnergyLossProcess, but it is overridden by in the derived classes
  - this may avoid unnecessary calls to AlongStepGetPh…Int… Lenght at each step
- G4VEmProcess and G4VEnergyLossProcess
  - looks like they originated from a common source and evolved
  - have quite similar functions some of which could be factorized

Other Aspects covered

- Looked at the impact of inlining and optimization (using gcc)
  - the optimized and inlined code is much faster (even by a factor of ~4 for the full SimplifiedCalo) from the non optimized/non inlined one
    - due to optimizations, even seemingly local code changes can modify the resulting final executable quite significantly (based on an inspection using GNU objdump)
- Done call chain and vectorization analysis

Various Observations

- Changing underlying data structures may have an impact bigger than the fraction of the CPU taken by the functions using them
- Some compilers produce code significantly (up to ~30%) faster compared to gcc
  - partially due to use of different math libraries
  - we may need to revisit this number for G4 v10 as many math functions changed
- puzzled by operator== and operator!=
  - they test for identity not equality (same address in memory vs. equal values)
    - quite common pattern not only in the EM code

Suggestions

- To eliminate unnecessary code maintenance and to (likely) improve performance
  - eliminate custom written copy constructors, assignment operators and destructors *when the compiler supplied ones are correct*
  - use the ?: (ternary) operator when possible
  - consider using Standard Library algorithms when available (usually more efficient)
    - e.g. std::lower_bound instead of hand written binary search (e.g. in the FindBinLocation functions
- Consider making G4PhysicsVector data members "read only" (and initialize them in the constructor) to help with the MT code (if one can accommodate data retrieval from a file in an appropriate way)
    - rewrite deriving (from G4PhysicsVector) G4PhysicsOrderedFreeVector to use another data member as it modifies its data members after creation

Suggestions - cont'd

- Institute routine code inspections
  - e.g. by a person chosen by the author of the code to be released
- Review the use and generation of random numbers; consider
  - routine use of the array interface
  - caching of the pointer to the engine
  - pregenerating the numbers (locally or globally)
  - adding API returning 1/randomnumber

Plans

- Collect more TAU results
  - to support some of the findings/conclusions
  - to analyze for other potential findings
- Quantify CPU effects of some of the code changes, done both by the authors or the reviewers, e.g.:
  - Inlining of G4PhysicsVector::FindBinLocation
  - Refactoring G4UniversalFluctuations::SampleFluctuations
  - double check using G4xyd helper in G4PhysicsVector
- Finish the review and write/deliver the report to the Electromagnetic Working Group/G4 Collaboration
  - patch release time scale?

Summary/Selected Findings

- Completed first pass through the EM code
  - some areas were even revisited for G4 version 10.
  - concentrated mainly on major features potentially impacting performance
- Gained about 3.5% in CPU by inlining G4Poisson & shorter functions in G4UrbanMscModel
- Other Findings
  - "Typo" like features in several places of the code
    - suggesting a need for routine code inspections?
  - Close similarities between G4VEmProcess and G4VEnergyLossProcess
  - Some G4VEnergyLossProcess classes should probably inherit from G4VDiscreteProcess (G4VEmProcess?)
  - Prevalent use of only single random number generator API via macro
    - using an array interface would decrease the number of calls and pave the way for future improvements; cashing the pointer to the engine would help as well