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

18th Geant4 Collaboration Meeting
Seville, September, 2013

Electromagnetic Code Review

presented by K. L. Genser

Outline

- Initial Scope
- Team
- Aspects covered so far
- Intermediate findings/suggestions
- Plans
- Summary

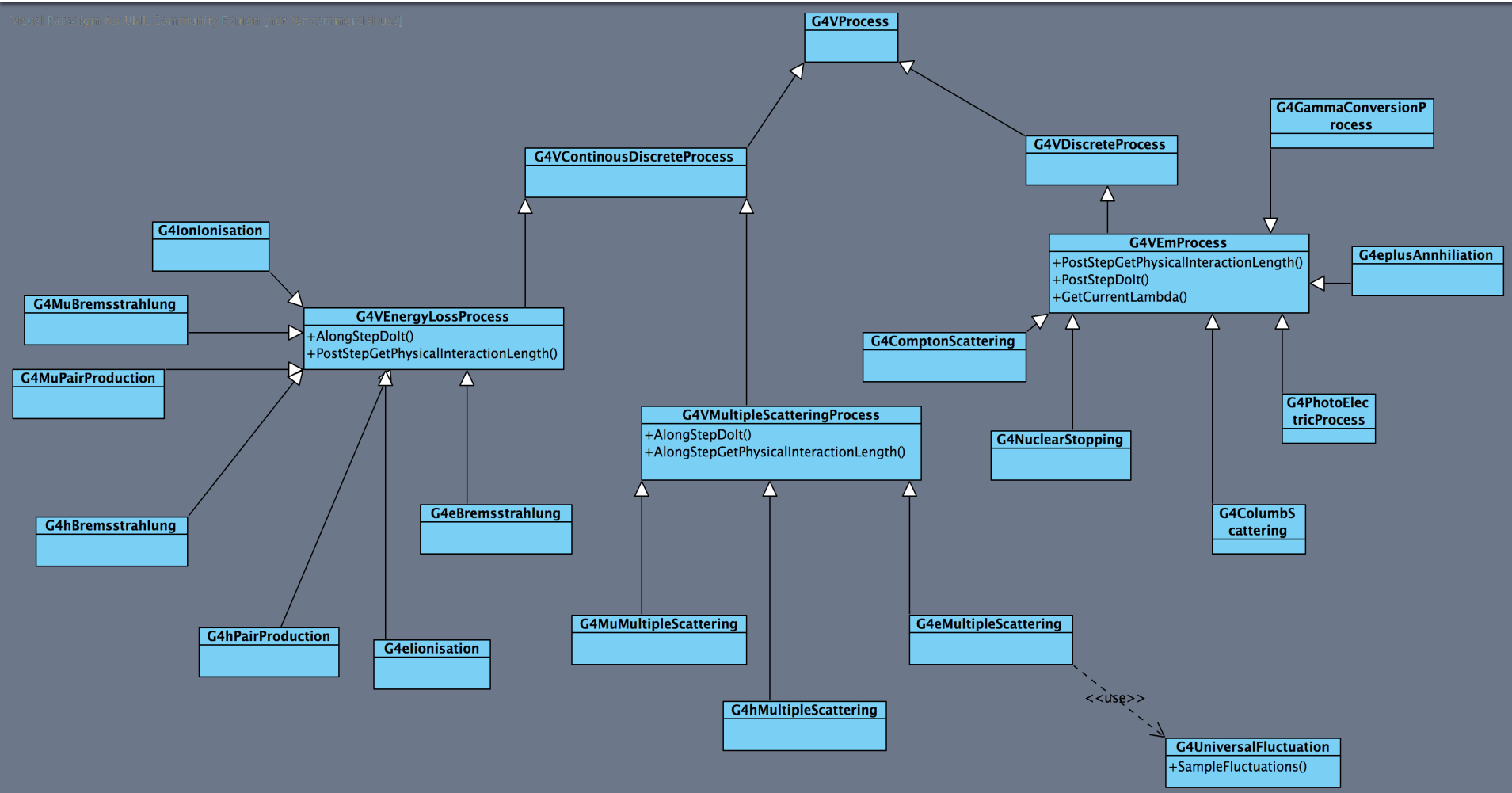
Scope and Initial Plans

- Review of ElectroMagnetic (EM) code has been in the plans of the Computing Performance Task Force for several years
  - It became possible thanks to US DOE support for a joint High Energy Physics (HEP) and Advanced Scientific Computing Research (ASCR) collaboration
- 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

Team

- Team members backgrounds and experiences cover various aspects of Geant4 and Computer Science
  - Geant4 itself
  - C++, source code analysis/transformation, performance tools, performance analysis, optimization
  - Profiling/Benchmarking
  - MultiThreading/GPUs/parallel code
- Mix of High Energy Physics and Computer Science backgrounds allows for interdisciplinary knowledge exchange and feedback also related to enhancement of code tuning and analysis tools

G4 EM Code Class Diagram



Areas covered so far

- Used v9.6.r07 as the basis of the current work
- G4PhysicsVector, esp. (Compute)Value
  - some underlying classes
- G4Physics2DVector
- Started looking at G4VEmProcess
- Settled on using SimplifiedCalo as the executable which performance we analyze to study the effects of the code transformations we undertake
  - it allows us to concentrate on the EM code related processes and to minimize the importance other factors
  - we have made sure that, while testing our changes, the final random number stayed the same after we modified the original code (mentioned in the later transparencies)

Code Transformations

- G4PhysicsVector (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 hand-coded binary search in G4LPhysicsFreeVector::FindBinLocation with std::lower_bound
  - replaced some ifs with the ?: (ternary) operator

Code Transformations cont'd

- The overall effect of transforming G4PhysicsVector 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 takes about 5% of the total execution time
  - the timing improvement did not occur in that function itself but in other areas, mainly in CLHEP::MTwistEngine::flat likely due to cache effects; under investigation

Code Transformations cont'd

- 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; under investigation
    - 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

Code Transformations cont'd

- Using random number generator array API
  - in G4UniversalFluctuations replaced

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

(G4UniformRand() is a macro expanding to:
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 tests, negligible effect in full tests, but allows for future optimizations

Other Aspects covered

- looked at the impact of inlining and optimization
  - 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 using GNU objdump)
  - more data at the Geant4 Profiling and Benchmarking page: https://oink.fnal.gov/perfanalysis/g4p
    (look for Other Test Results, r07)
- done call chain and vectorization analysis

Current Suggestions

- Transform G4PhysicsVector to use a helper class (e.g. G4xyd) with three G4double data members
  - make the (G4xyd) G4PhysicsVector data members "read only" (and initialize them in the constructor) to help with the MT code
    - rewrite deriving (from G4PhysicsVector) G4PhysicsOrderedFreeVector to use another data member as it modifies its data members after creation
- 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
- In G4Physics2DVector::FindBinLocation use std::lower_bound

Observations/Questions

- Changing underlying data structures may have an impact bigger than the fraction of the CPU taken by the functions using them
- There are compilers which produce code significantly (up to ~30%) faster compared to gcc
  - partially due to use of different math libraries
- 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
    - what was the original idea behind it?

Plans

- Investigate the specific reasons why changing the G4PhysicsVector and G4Physics2DVector internal data containers impacts the performance of the code so significantly
  - the information may be important in other code areas
- Continue to review compute intensive EM functions esp. in classes deriving from G4VContinuousDiscreteProcess
- Investigate shortening of the call/inheritance hierarchies
- Finish the review and write a report

Summary

- Formed an interlaboratory/interdisciplinary team to review ElectroMagnetic(EM) code
  - providing feedback to code performance analysis tools developers helps to improve those tools and will eventually help to improve Geant4
- Reviewed some aspects of G4PhysicsVector, G4Physics2DVector classes and started looking at G4VEmProcess
  - Arrived at preliminary conclusions regarding G4PhysicsVector and G4UniversalFluctuations
    - some of which may be applicable elsewhere, esp. the ones regarding the use of compiler supplied copy constructors and assignment operators and the use of Standard Library algorithms whenever possible
  - Still investigating the importance of the data structures esp. in G4Physics2DVector
- Plan to continue the review
  - further analysis the containers of G4Physics2DVector and compute intensive EM functions esp. in classes deriving from G4VContinuousDiscreteProcess
  - investigations of shortening of the call/inheritance hierarchies
- Feedback is more than welcome; Any suggestions regarding the plans?

Supplementary Slides

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, G4UrbanMscModel95, (or G4UrbanMscModel in the newer version) esp. ComputeGeomPathLength, ComputeTruePathLengthLimit, SampleCosineTheta, SampleScattering

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 profiler for standard profiling/benchmarking; also using HPCToolkit, TAU (Tuning and Analysis Utilities) Performance System and Open|SpeedShop
- More info at https://oink.fnal.gov/perfanalysis/g4p/admin/task.html