Modernizing the ATLAS Simulation Infrastructure

Andrea Di Simone
University of Freiburg (DE)

On behalf of the ATLAS collaboration

Outline

- ATLAS Simulation
- New challenges and the modernization of the legacy code
  - MCTruth
  - Geometry handling
  - Parallel processing
  - Performance optimization
  - Code packaging and distribution
- Conclusions

ATLAS simulation

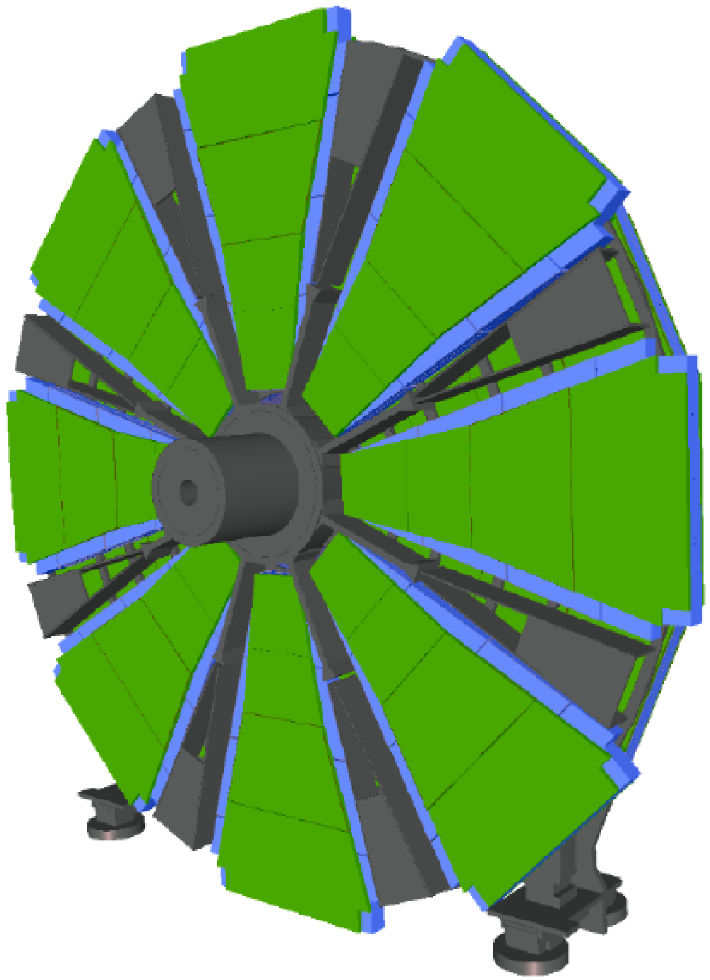
- Fully integrated into the rest of the offline software
  - Allows to run full simulation (Geant4), as well as fast, parameterized simulations
    - Full simulation uses Geant4 to simulate the propagation of particles through the detector
  - Hybrid full/fast simulation is also supported
- Used to simulate events needed for the design of the experiment and for data analysis
- ~50 billion events simulated, ranging from cosmic rays to quantum black-holes, from test-beams to collisions
- Stable and robust operation using distributed resources

Simulation framework

- Former full-sim implementation based on a custom framework (FADS)
  - Developed in the early 2000s
  - Provides access to all basic simulation functionalities
    - Geometry description
    - User action definition
    - Sensitive detector handling
    - MC Truth handling
    - Magnetic field
  - Originally designed to run stand-alone simulation jobs on top of Geant4
    - Adapted in the following years to allow better interoperability with the rest of the (Gaudi-based) ATLAS offline software
- Work currently ongoing to move away from FADS
  - Direct use of Gaudi components whenever possible results in a more streamlined and maintainable code
  - Configuration layer (python) is also being re-designed for improved robustness and flexibility

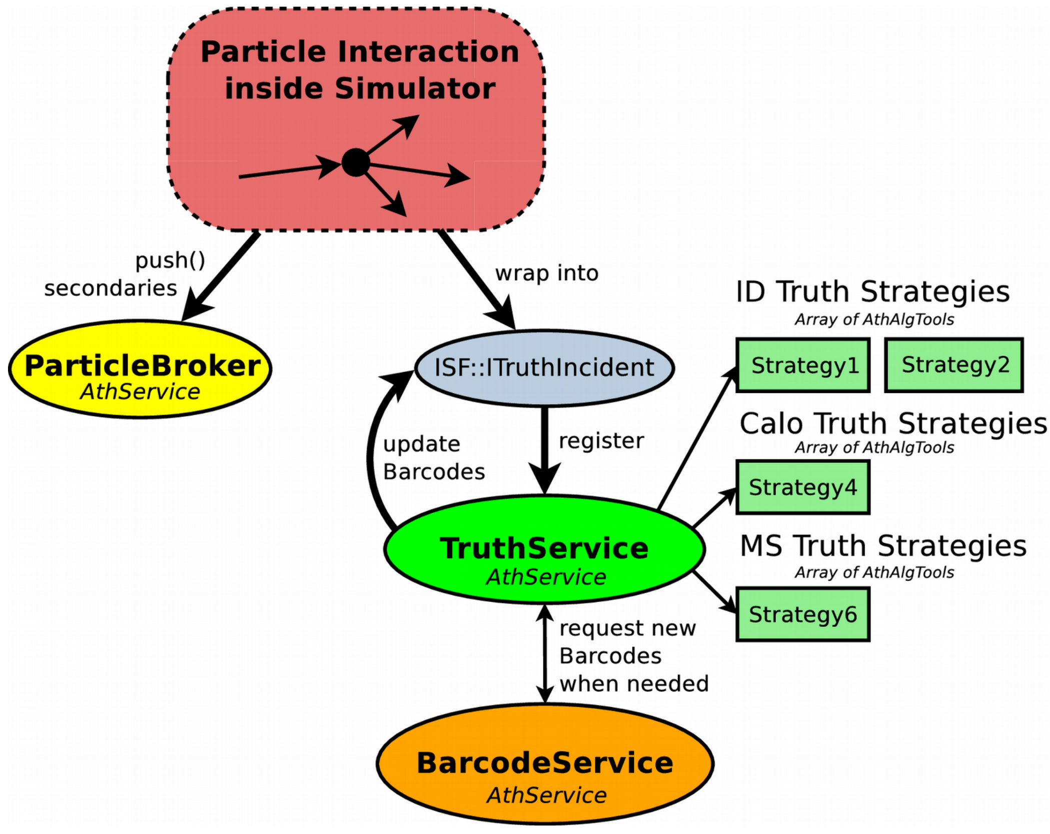
Geometry description

- Current geometry built at runtime starting from data stored in a dedicated database
  - Resulting geometry is shared between all ATLAS software (simulation/digitization/reconstruction)
  - Very robust approach, designed to serve the needs of large-scale simulation with a frozen detector layout
- Work for the LHC/ATLAS upgrade, on the other hand, needs a somewhat more flexible approach
  - R&D for the new detectors requires rapid turn-around and possibility to quickly implement modifications
- The geometry handling has been re-designed, based on modular, recursive components to define the volume hierarchy
  - Modifying a single subdetector without touching the rest is now relatively easy
  - More options are available for providing the detector description itself, including xml-based formats (AGDD, DD4hep)



MCTruth handling

- Simulation adds entries to the MCTruth originating from the event generators
  - Too many secondaries are created to store everything, hence the filtering/handling itself is far from being trivial
- For historical reasons, different “flavours” of simulation (e.g. fast and full) do not share the same code for MCTruth
- Effort ongoing to replace the full-sim-specific code with a more generic one, usable for different fast simulations
- Migration being done in different steps, making sure that results remain bit-wise identical at every step

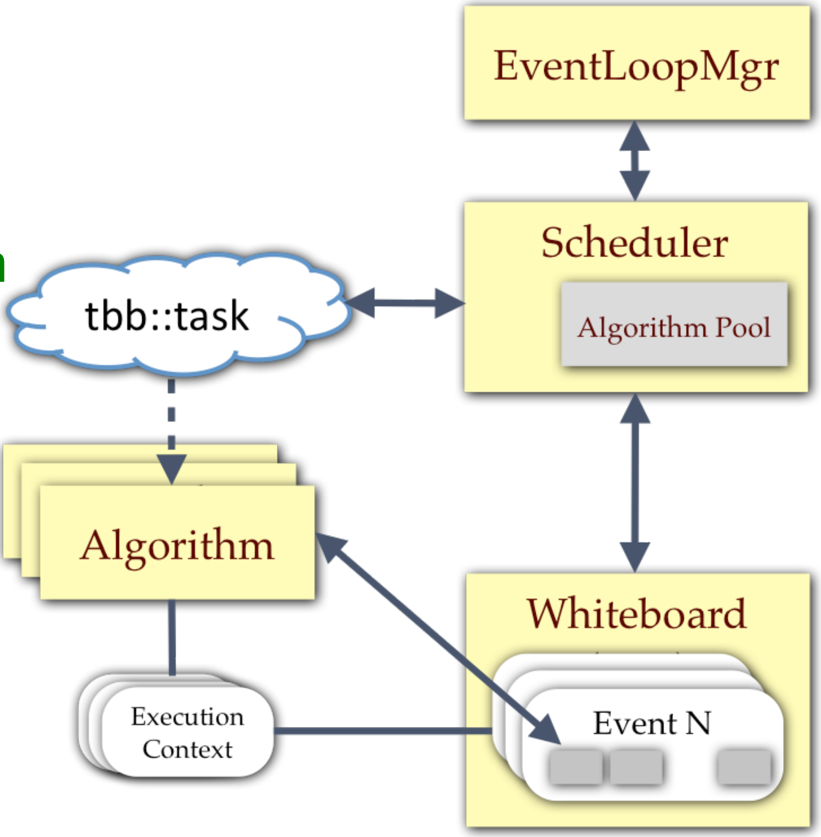


The challenge of HPC

- With the increase of integrated/instantaneous luminosity, the experiment has even more pressing needs in terms of number of events to be simulated
- On the other hand, the hardware standards have been moving very strongly towards massively parallel architectures
- Most of our code-base is just not ready for parallel execution
- A massive effort is currently ongoing to rewrite/restructure/repackage all the simulation code
    - part of the more general attempt to use GaudiHive as the base framework for the ATLAS offline software

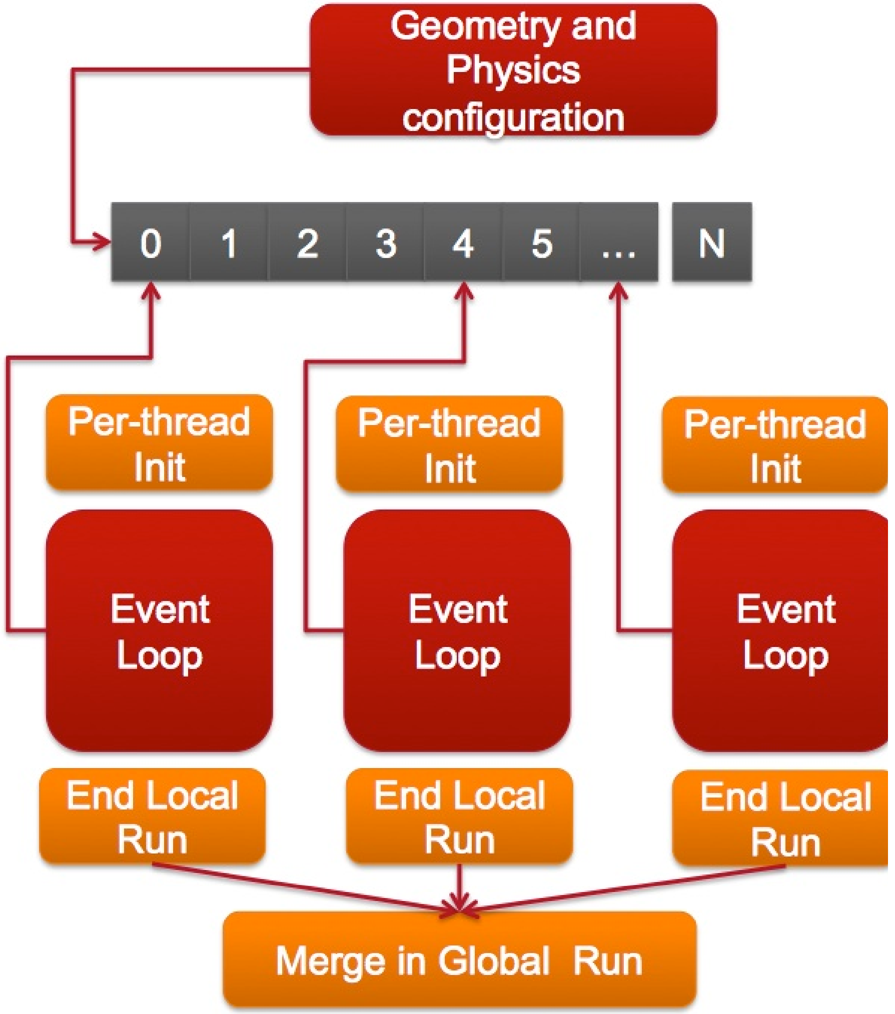
GaudiHive

- A multi-threaded, concurrent-execution extension to Gaudi
  - Concurrency model based on TBB
- Scheduling is driven by data-flow
  - Data dependencies of algorithms are explicitly declared
  - Availability of input data triggers execution of Algorithms
- Events processed in multiple threads
  - Multiple algorithms and events can be executed at the same time
- Several parts of the ATLAS offline software being prepared to run on top of GaudiHive
  - See talk by C. Leggett on Monday morning session



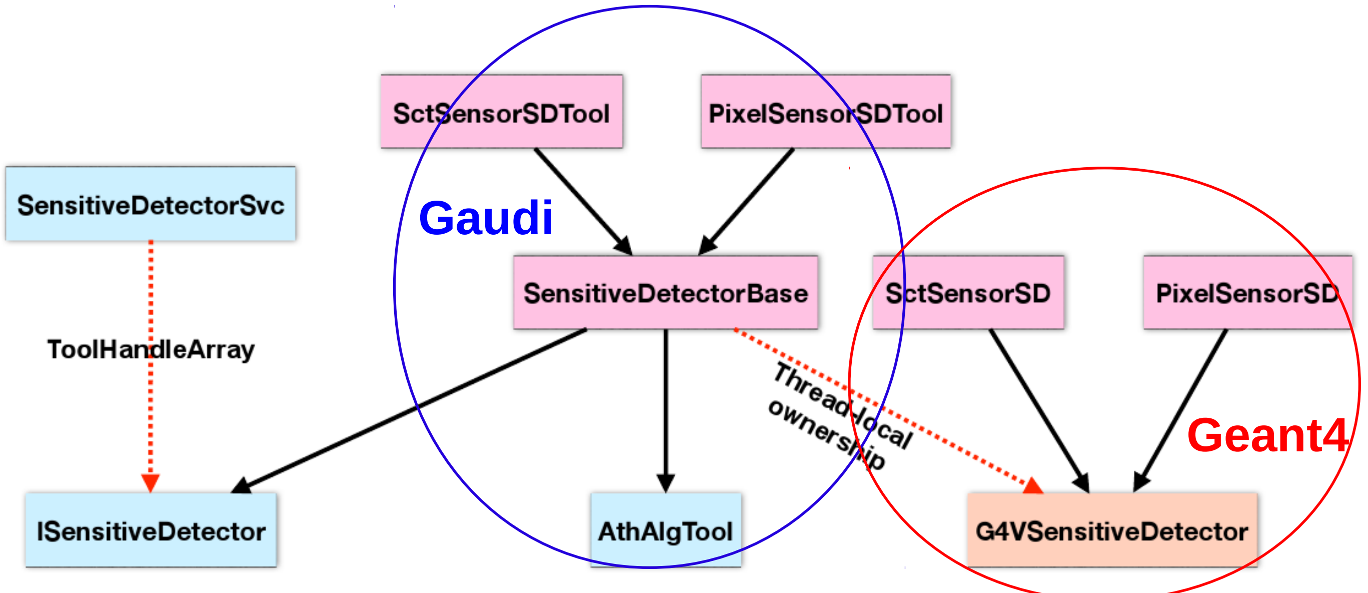
Geant4MT

- Full simulation is, however, a special case
  - Its “core” functionality is actually living in an external *third-party* package (Geant4)
- Geant4 has its own approach to parallel processing
  - Master-slave concurrency model, using pthreads
  - Provides event-level parallelism
- Thread safety achieved using thread-local storage



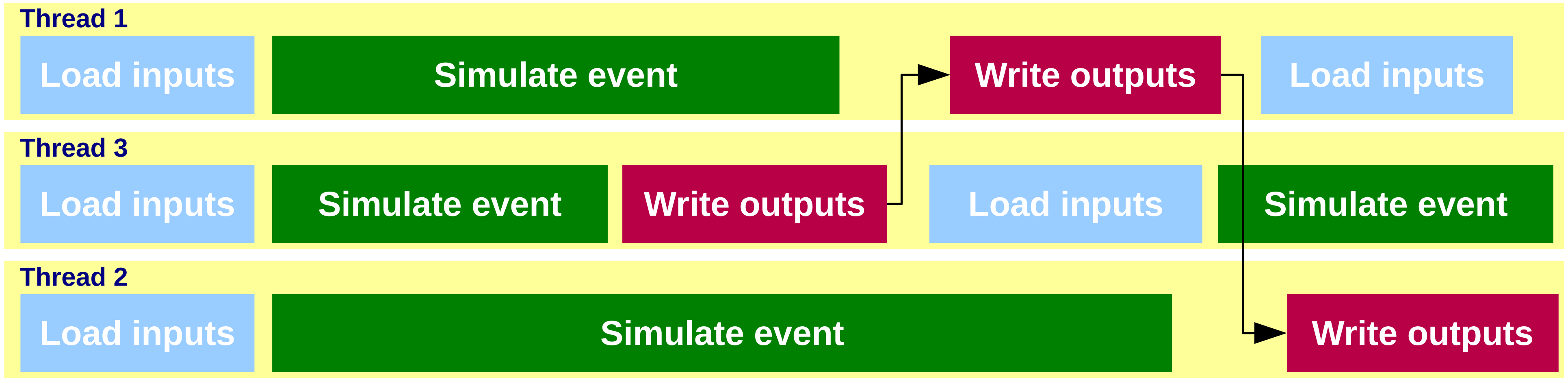
G4Hive

- Main Geant4MT components ***must*** be thread-local
  - Run Managers, Sensitive Detectors, User Actions, Magnetic Field
- However, GaudiHive provides task locality, not thread locality
  - Cannot easily pin a Gaudi component to a specific thread
- Must decouple the Gaudi components from the Geant4 core functionality
- Current design (and implementation): Gaudi components own the corresponding thread-local G4 objects



Implementation overview

- Initialization is very tricky: G4 requires that thread-local objects are initialized in their threads at the right time
  - Required tuning of the thread-initialization mechanism in GaudiHive
- The whiteboard is populated with initial data and the dependency chain is bootstrapped
- Dedicated algorithms take care of the simulation itself, forwarding the processing of an event to the Geant4 components when needed
- At the end of every event, the hits produced are written (serially) to the output file

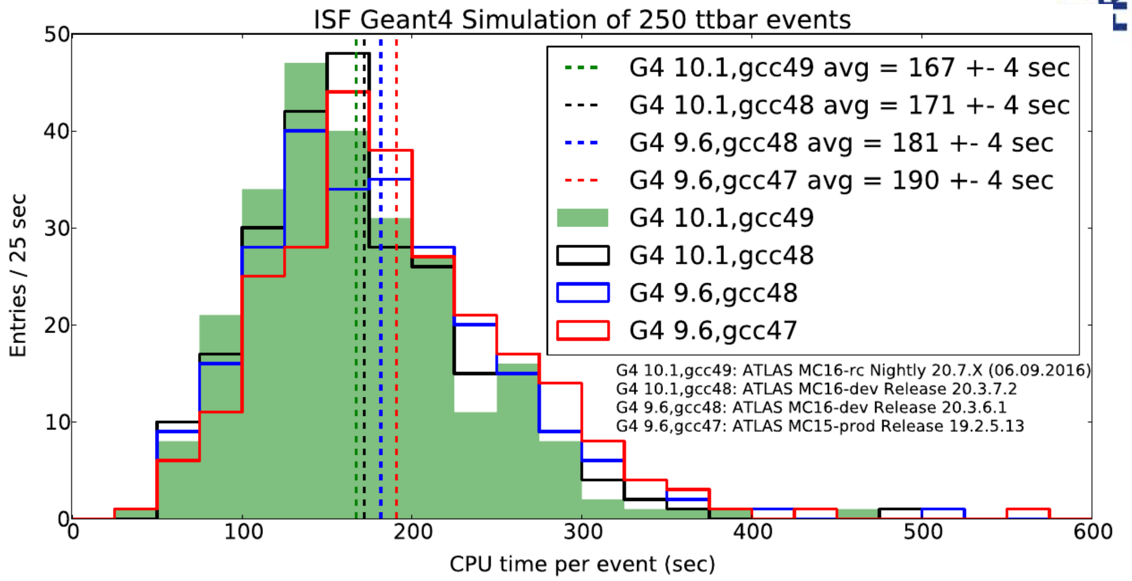


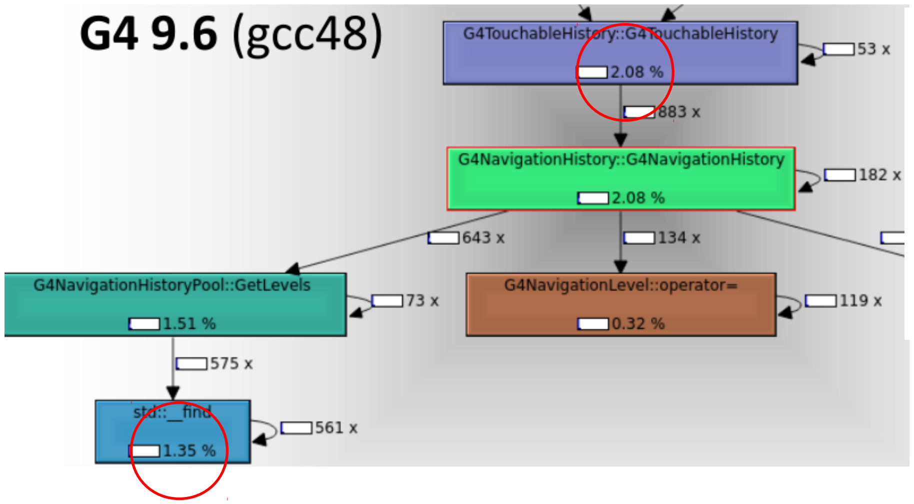
Current Status

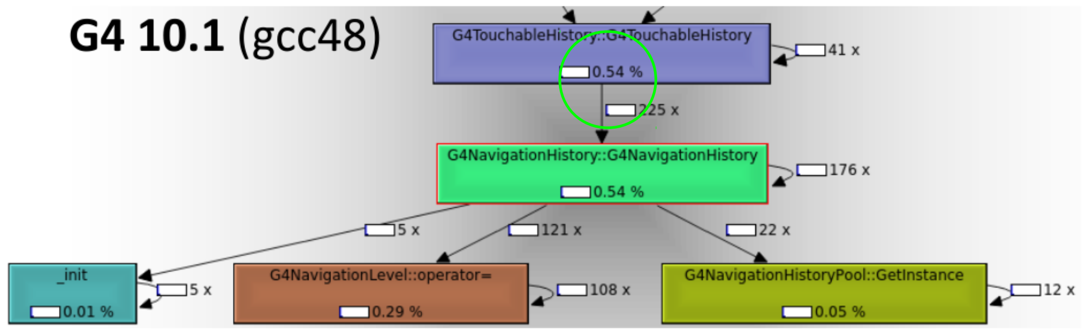
- Most of the functionality has been migrated away from FADS and to the new Hive-friendly architecture
  - Sensitive detectors
  - Fast simulations
  - User Actions
  - Physics processes
  - Geometry
  - Magnetic field
- Migration involved modifications to most of our code-base
  - A perfect opportunity for an extensive code review
- Interventions that could potentially change the physics output were clearly identified and separated from the purely technical redesign
  - Very thorough step-by-step validation
- Last big chunks of code being finalized are
  - MCTruth configuration
  - Some Calorimeter specific code
- See talk by S. Farrell on Thursday for more details and results

Performance monitoring

- Detailed profiling being run at every new G4/Athena release
  - Example: impressive 15% speedup between two production campaigns, due to
    - New G4 version
    - New gcc version
    - ATLAS code cleanup




G4 9.6 (gcc48)
G4TouchableHistory::G4TouchableHistory
2.08 %
53 x
883 x
G4NavigationHistory::G4NavigationHistory
2.08 %
182 x
643 x
134 x
G4NavigationHistoryPool::GetLevels
1.51 %
73 x
G4NavigationLevel::operator=
0.32 %
119 x
575 x
std::__find
1.35 %
561 x


G4 10.1 (gcc48)
G4TouchableHistory::G4TouchableHistory
0.54 %
41 x
225 x
G4NavigationHistory::G4NavigationHistory
0.54 %
176 x
5 x
121 x
22 x
_init
0.01 %
5 x
G4NavigationLevel::operator=
0.29 %
108 x
G4NavigationHistoryPool::GetInstance
0.05 %
12 x

Packaging and distribution

- Full ATLAS offline release has about 2200 packages, and uses about 16GB of disk space
- Attempt to separate the simulation-specific code and re-package it into a smaller, more portable Athena release: ***AthSimulationBase***
  - Could be used to run a G4 simulation on systems with very tight disk/memory requirements
  - Current implementation has about 390 packages, using 3.7GB on disk
    - ATLAS code + G4 libraries is about 270MB, G4 input data is 1.6GB, the rest due to other external dependencies
- A smaller code-base with fewer dependencies helps in porting the simulation to new architectures (such as ARM)
- The factorization of the code also helps to design and maintain cleaner and simpler dependency graphs
- An AthSimulationBase release already validated for official production
- A CMake build is also available, whose nightlies rebuild G4 based on the G4 git repository
  - A very quick way to assess impact on ATLAS simulation of new G4 developments

Conclusions

- The ATLAS simulation code has been assisting detector design and data analysis since the very inception of the project
- In its current implementation, it is being extensively run on distributed resources to fulfill the needs of an ever growing number of analyses
- The large amount of legacy code is getting more and more challenging to maintain
- New challenges have pushed the experiment to embark on a major code modernization effort
  - Main focus on increased flexibility, performance optimization, parallel processing
- Migration carried on in steps, every step thoroughly validated against the old code
- The status today is an almost complete detector simulation, capable of running in multi-thread mode
  - The few missing parts of code are being worked on as we speak