https://gitlab.cern.ch/G4MSBG

Medical Benchmarking Testing Suite

Cmake/ctest based integration

Andrea Dotti (adotti@slac.stanford.edu) ; SLAC/SD/EPP/Computing

for the Geant4 Medical Simulation Benchmarking Group

Geant4 21st Collaboration Meeting – Ferrara, 12-16 September 2016

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Note

This talk introduces the technical aspects of the testing and integration suite for the Geant4 Medical Simulation Benchmarking Group

A dedicated parallel session (2A) discusses the physics tests being implemented in detail

Integration

SLAC

G4MSBG: a set of independent Geant4 applications to test and validate specific aspects of simulations for medical users (e.g. Bragg peak)

- Focus on applications that allow to compare with real data

Applications are developed by different people following a “style” (e.g. how analysis is done and how results are presented) that is tailored to the specific problem

However we would like to allow anybody (from the G4MSBG group) to run any of the validations in a coherent system

- When possible (automatic) statistical testing and comparison with data is encouraged to help understanding the quality of simulation by a non-expert

Choice of tools 1

SLAC

Project external to Geant4 but aiming at tight integration

Code work model: I'm proposing a simplified git-flow (not yet enforced), discussion?

Code repository: gitlab @ CERN, with a git group G4MSBG owning the repository

Choice of tools 2

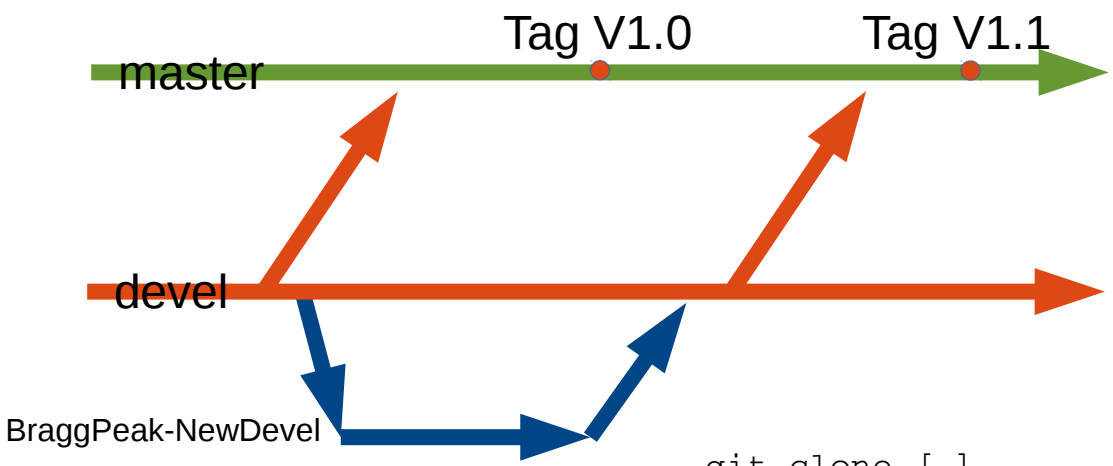
SLAC

CMake as configuration tool (to simplify integration with Geant4)

CTest as testing tool

- **Pro**: Testing is very simple, same platform as Geant4, possible to use Geant4 cdash for web-presentation (if Gunter agrees)
- **Cons**: Not easy to use distribute testing on more than one node (that may be necessary if number of tests continues to grow)

Proposed Development Work-flow



Developments never go here
Checking out master always compiles and runs latest version.
Official “version tags” are made here.

Developments go here. Used for integration Testing.
You should branch your own development Branch for specific developments.
Merging to master done by coordinators

```
git clone […]
git checkout devel
git checkout -b BraggPeak-NewDevel
[… new develops …]
git add […] && git commit -m “comment”
git checkout devel
git merge –no-ff BraggPeak-NewDevel
git branch -d BraggPeak-NewDevel
git push
```

Packages Organization

SLAC

```
G4MSBG
      CmakeLists.txt
      cmake
      BraggPeak
          CMakeLists.txt
          CTestDefinitions.txt
      attenuation
      DPK
      ...
```

Pre-requisites, configuration and compilation

It requires Geant4 already compiled and installed (via cmake)

Some applications depend on ROOT

For testing with ctest (optional):

- Requires StatTest application (see my talk at this session)
  - That by itself requires python and ROOT's python support

```
git clone […] && mkdir build && cd build
cmake -DGeant4_DIR=<...> [-DG4MSBG_ENABLE_TESTING=ON] ../G4MSBG
make -j<N>
```

It is possible to compile only a single application (see content of generated Makefile for list of targets):

```
make -j<N> BraggPeak
```

If you are not interested in ctest integration you can also configure/compile a single application with:

```
git clone […] && mkdir build-BraggPeak && cd build-BraggPeak
cmake -DGeant4_DIR=<...> [-DG4MSBG_ENABLE_TESTING=ON] ../G4MSBG/PraggPeak
make -j<N>
```

Testing with ctest: introduction

SLAC

Based on the code I've received and the interaction with developers I've created, for each application a set of "ctests" to perform physics validation.

- In many cases this meant transforming a bash script to a ctest one
- Each run-time configuration corresponds to a separate ctest
  - e.g. for BraggPeak application, each input macro is a separate ctest; for TestDEDX2 each combination of {physicslist,primary,material} is a ctest

Notable difference: since w/ ctest we run tests in parallel one important step is to (automatically) generate an output directory structure where each test can run in isolation

- This directory structure (and possibly generation of macro files from templates) is created when "cmake" is executed

Testing with ctest: test definition

SLAC

Testing has to be explicitly enabled when executing cmake

Tests are defined in `CTestDefinitions.txt` script provided with each application
Type: **`ctest -N`** to see the list of all tests.
Type: **`ctest -N -R <name>`** to see the list of all tests containing the string <name> (ctest –help for the list of all options).
Test names clearly identify application and test condition, for example:

- `Ctest -N -R BraggPeak`
  - `BraggPeak_200MeV.INCL`
  - `BraggPeak_200MeV.INCL-checkOutput`
  - `BraggPeak_200MeV.QMD`
  - `BraggPeak_200MeV.QMD-checkOutput`
  - `[...]`

Type: **`ctest -j <N> [-R <name>]`** to execute tests
Tests will be executed in parallel (N jobs), but dependencies will be honored when specified (e.g. `BraggPeak_200MeV.QMD-checkOutput` is executed iff `BraggPeak_200MeV.QMD` has been already executed succesfully)

Testing with ctest: post-processing

SLAC

Tests are divided in two phases:

- **Step 1**: run Geant4 application. An output file (e.g. histogram file) is produced
- **Step 2**: run post-processing (e.g. create image) or statistical testing on the output of Step 1

Step 2 appears as a separate ctest, with name as in Step 1 and postfix “-checkOutput” or “-postProcessing*”

Testing with ctest: StatTest integration

SLAC

When applicable Pedro has provided the script to run the StatTest utility and perform regression testing

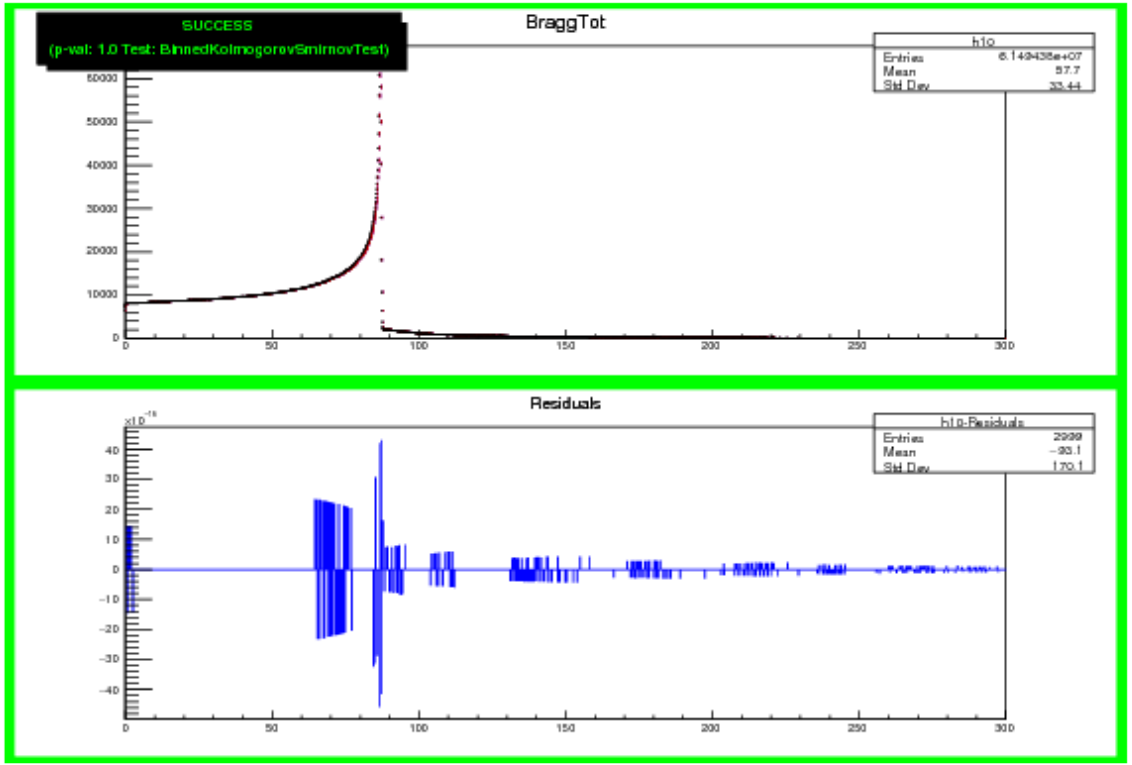
- Introduced in ctest accordingly

Idea is to be able to compare two G4 versions and perform statistical tests

Need to decide where to store references: proposal is to store them to a web-server at CERN dedicated for this purpose (so StatTest can found these references automatically), if Gunter agrees

- It means that developers will have to provide an updated reference from time to time

Example: output of ctest for BraggPeak 200MeV.INCL



Red/Blue histograms are the two versions being compared
P-val and test info are printed
Bottom histogram is residuals plot and border is color coded:
Green=OK, Red=BAD, Yellow=NEEDATTANTION

Status

Application	Total Number of tests	StatTest ready
attenuation	10	Yes
BraggPeak	12	Yes
DPK	41	Yes
ElecBackScatBench	352	No, but summary plots created
TestDeDX2	183	No, but summary graphs created
Total	**598**	

FragBenchmarking received but not yet integrated

Conclusions

SLAC

G4MSBG test suite is in gitlab@CERN
Integration with cmake and ctest underway

5 (6) applications received so far and ctest scripts prepared

Note: currently running the complete test-suite will take not less than 12 hours on a 12 cores server, with the growing number of tests we may need to dedicate some resources to testing if we want to perform regular testing (I think I can guarantee these at SLAC)

I think the same model could be also used by other Geant4 group of users, we may think about creating a public web-page with instructions and script templates