Geant4 Collaboration Meeting 2023

Geant4.jl vs. geant4_pybind (vs. C++ geant4)

Pere Mato / CERN

https://github.com/JuliaHEP/Geant4.jl
https://github.com/HaarigerHarald/geant4_pybind

Motivation

- geant4_pybind
  - provide complete Python bindings for Geant4, such that users can write applications only with Python (not the case for g4python)
- Geant4.jl
  - evaluate the Julia interoperability with existing large C++ packages
  - provide a complete Julia bindings for Geant4, such that applications can be written only with Julia

Comparison Criteria Set

1. Ease of Installation
2. Friendliness of API
3. Interactivity
4. Running Speed
5. MT Support
6. Visualisation/Analysis capabilities

1. Ease of Installation

- C++
  - Build from sources is the most common installation method
  - Package managers such as ‘conda’ also provide Geant4
  - User applications are built with CMake against Geant4 installations
- Py
  - ‘pip install’ can provide pybind wrappers linked statically with G4 libraries
  - User can also locally build the wrappers extension, but this needs an installation of Geant4
- JL
  - Everything is managed by the native Pkg package manager of Julia ‘Pkg.add(“Geant4”)’
  - Geant4.jl (Julia) --> Geant4_julia_jll (wrappers) --> Geant4_jll (g4 libraries)

2. Friendliness of API

- C++
  - The API is rather complex requiring the user to provide classes that inherit from given base classes and implement specific methods to provide all that the toolkit needs
  - The order in which operations are done is critical (non declarative)
  - MT adds non negligible additional burden to the user
- Py
  - Better since it requires less from the user (no header files, no need to build libraries) but follows exactly the same API as the C++
- Jl
  - New minimalistic API (no boilerplate), hiding the details (doing the necessary at the right time) and simplifying the multi-threading (e.g. per-thread calls and thread-local instances)

3. Interactivity

- C++
  - Limited capabilities
  - Either using the UI or the GUI interfaces, but only being able to interact with what has been anticipated by the developers
- Py
  - Emulates the C++ user interface (UI) capabilities despite having the powerful Python REPL
  - Jupiter notebooks fully supported
- Jl
  - The Julia REPL can be used to interact with any C++ wrapped class and with the new Julia interface. No need to implement any G4UImessenger.
  - Jupiter and Pluto notebooks fully supported

4. Running Speed

- C++
  - Reference point for both serial and MT
- Py
  - G4 engine code the same as C++
  - User actions/sensitive detectors implemented in Python, which is many factors slower than C++
- Jl
  - Longer initial startup time (because of JIT)
  - For long runs should be as fast as C++ (within few %)

	B2a (C++)	B2a.jl	B2a.py
events = 1	0.9 s	6 s	1 s
events =100k	106 s	109 s	176 s
events =100k (MT)	23 s	27 s	--

- Simple benchmark of B2a example
  - with protons @ 3 GeV
  - running on a Mac-mini with the M1 processor (8 cores = 4 performance and 4 efficiency)
- C++ and Julia are basically identical taking the initial overhead (serial) into account

5. MT Support

- C++
  - Clear recipes to be followed to convert a Serial application into MT
  - Complexity is not completely hidden to the user
  - User needs to adapt the code in detector construction, user actions, analysis, etc.
- Py
  - Not supported (Python Global Interpreter Lock - GIL)
- Jl
  - Single parameter controls the running mode (‘nthreads’) in G4JLApplication
  - User needs only to provide a ‘reduce’ function to sum partial simulation custom data

6. Visualisation/Analysis capabilities

- C++
  - Geometry and event display capabilities useful for debugging. Complex to install and use.
  - Limited capabilities for analysis. Typically data is extracted, and saved to be used offline with other tools (e.g. ROOT, Python, etc.)
- Py
  - Using the C++ geometry and event display capabilities
  - Full Python ecosystem available for analysis
- Jl
  - Julia visualisation packages are extremely simple to integrate (e.g. Makie.jl, Plots.jl)
  - Full Julia ecosystem available for analysis
  - Online analysis (e.g. active filtering) in addition to offline analysis

Summary

	C++	PyBind	Julia
Ease of Installation	Fair	Very Good	Very Good
Friendliness of API	Bad	Fair	Good
Interactivity	Fair	Very Good	Very Good
Running Speed	Very Good	Bad	Very Good
MT support	Good	Very Bad	Very Good
Visualization / Analysis	Fair	Good	Good