Conan2 based builds for CORSIKA8

A. Augusto Alves Jr

Presented at CORSIKA development meeting - KIT, Karlshuhe

June 20 - 2024

Recap: CORSIKA 8 build

CORSIKA 8 is basically a header-only framework which orchestrates a workflow involving a collections of components to simulate particles cascades in material media.

- Components: modules, libraries and data.
- Different computing languages: C++, C, FORTRAN, Python.
- Most of these these components needs to be built ahead usage.
- Many direct and indirect dependences, with different building procedures.

Most of the complexity to build CORSIKA 8 comes from the management of dependencies.

Dependencies management: enters Conan

Conan is an open source, decentralized and multi-platform package manager to create and share native binaries.

- CORSIKA 8 users are "encouraged" to use Conan to install the necessary dependencies.
- Actually, currently, Conan is "hard-coded" in CORSIKA 8.
- The CMake scripts, which build and install CORSIKA 8 components, search specifically for packages installed via Conan.
- It is hardly satisfactory, since many of such packages are also available as system packages. It is also hard to build CORSIKA 8 with an user-tuned package.
- This architecture defeats most of the flexibility provided by CMake to build different configurations and promotes code bloating.

Currently, CORSIKA 8 build process requires Conan-1.X.Y series, which is now EOL.

Ideally...

Pretty simple:

- CORSIKA 8 should not care if a dependency was installed via Conan, some other package manager, or even is provided by the system. Basically, CMake scripts in CORSIKA 8 should be agnostic about it.

This goal can not be achievable using Conan-1.X.Y. series, but Conan 2.X.Y series is fully compliant with this strategy.

Conan-2

Conan-2 provides different tools to integrate with CMake in a transparent way. Using these tools, the project CMakeLists.txt file does not need to be aware of Conan at all.

- CMakeDeps: responsible for generating the CMake configuration files (`<name>Config.cmake`) for all the required dependencies. Examples: `BoostConfig.cmake`, `Catch2Config.cmake`...
- CMakeToolchain: generates all the information needed for CMake to build the packages according to the information passed to Conan about things like the operating system, the compiler to use, architecture, etc.
- Python based recipes.

Integration into CORSIKA 8

Omitting implementation details, basically:

- As configured in the recipe (`corsika/conanfile.py`), Conan-2 will download, build and install the required dependencies.
- It will also generate corresponding configuration files for CMake to find the packages and store all it in locally created `corsika/conan_cmake` directory. This directory is on `.gitignore` file.
- Once the directory `corsika/conan_cmake` exists and is populated, it is added to `CMAKE_MODULE_PATH` in the COSIKA 8's `CMakeLists.txt` , via definiton of the variable `CONAN_CMAKE_DIR` .
- Before looking-up for configurations required by `find_package(...)` commands on system-wide defined paths, CMake will consume what is on `corsika/conan_cmake` , and then pick-up the packages installed by Conan-2.

Integration into CORSIKA 8

Two scripts manage the installation of the packages via Conan-2 and the configuration of CORSIKA 8 for building and installation:

- `conan-install.sh`: It will invoke the Conan-2 commands to install the dependencies in `\$CONAN_HOME`, generate the CMake tool-chain and configuration files, places it in the `corsika/conan_cmake` directory and generate the CORSIKA 8 configuration script `corsika-cmake.sh`.
- `corsika-cmake.sh`: It substitutes the conventional `cmake ..` command. It will emit the correctly configured instructions with the paths and variables needed to activate the Conan-2 integration. After this... `make -jN`.

Integration into CORSIKA 8: `conan-install.sh --help`

```

|--------------------------------------------------|
|-----------------[ CORSIKA 8 ]---------------------|
|-----[ CONAN2 DEPENDENCIES INSTALL SCRIPT ]------- |
|--------------------------------------------------|
|-------------------- BEGIN -----------------------|

[ conan-install | info > This script is located at the directory:  <...>/corsika

Usage:./corsika/conan-install.sh options [parameters]

Options:
 -s or --source-directory:
 Corsika 8 download directory, which contains the 'conanfile.py' recipe. Default is the current directory.
 -d or --debug:
 Specify 'Debug' as build type for the installed dependences. This should be matched when building CORSIKA 8.
 -r or --release:
 Specify 'Release' as build type for the installed dependences. This should be matched when building CORSIKA 8.
 -rd or --release-with-debug:
 Specify 'RelWithDebInfo' as build type for the installed dependences. This should be matched when building CORSIKA 8.

Example: ./conan-install.sh --source-directory /some_path/corsika --debug
 -h or --help:
 Prints this message.
```

Integration into CORSIKA 8: `corsika-cmake.sh --help`

```
|-------------------------------------------------|
|-----------------[ CORSIKA 8 ]-------------------|
|----------[ CMAKE CONFIGURATION SCRIPT ]---------- |
|-------------------------------------------------|
|-------------------- BEGIN ----------------------|

Usage: <...>/corsika/corsika-cmake.sh options [parameters]

Options:
 -c or --cmake-flags:
 Additional flags and settings to cmake base command. Default is empty string.

Example: ./corsika-cmake.sh --cmake-flags '-DUSE_Pythia8_C8=C8'

Note: the source directory (the one containing CMakeLists.txt),
CMAKE_BUILD_TYPE, CMAKE_POLICY_DEFAULT_CMP0091 and
CMAKE_TOOLCHAIN_FILE are already set. Do not repeat them.

 -h or --help:
 Prints this message.
```

Integration into CORSIKA 8: Comments

- The build type of the dependencies installed by Conan-2 and CORSIKA 8 have to match.
- Three build types available: `Debug`, `Release` and `RelWithDebInfo`.
- Using `conanfile.py` instead of `conanfile.txt` allows to solve version conflicts by forcing one specific version. Practical, but dangerous.
- Mybe We should keep all dependencies updated to avoid major re-factories.

Dependencies: version updates

Package	Master Branch	Conan-2 Branch	Intervention
Catch2	v2.13.8	v3.6.0	Code and CMakelists.txt
Spdlog	v1.9.2	v1.14.1	"
BZip2	(not listed)	v1.0.8	CMakelists.txt
Boost	v1.78.0	v1.85.0	"
Eigen	v3.3.8	v3.4.0	"
ZLib	v1.2.13	v1.3.1	"
yaml-cpp	v0.7.0	v0.8.0	"
cli11	v1.9.1	v1.9.1	"
Arrow	v10.0.0	v16.1.0	"
Proposal	v7.6.2	v7.6.2	"

Dependencies: version updates

- Catch2. Fix scope problems of some operators and functors. Catch2 is multi-header since a while, so `#include<catch/catch2.hpp>` directives needed revision.
- Spdlog. Required implementation of some formatters for some specific types. A new header was introduced to handle this: `corsika/detail/framework/core/SpdlogSpecializations.inl`.
- `*.inc` are now produced at configuration time, instead of building time.
- `corsika/cmake_conan` directory is installed at `lib/cmake/dependencies`.

Note: No physics/math sensitive code was touched.

Conclusions, comments and perspectives

- Conan-2 has been integrated with Corsika 8 in a transparent and decoupled way.
- Work is done at branch `conan2_cmake_building`
- Version of several dependencies has been updated.
- CI related scripts/configurations have been updated.
- All tests are passing.
- Personally, I think we can safely pass to Conan-2 based builds, but I would recommend some code review.
- Discuss our policy for dependency updates.
- Consider alternative package managers: vcpkg, spack.

Backup Slides

CMakeLists.txt

```
if(DEFINED CONAN_CMAKE_DIR)
        list(APPEND CMAKE_MODULE_PATH "${CONAN_CMAKE_DIR}")
endif(DEFINED CONAN_CMAKE_DIR)
...
find_package(Boost COMPONENTS filesystem REQUIRED)
find_package(CLI11 REQUIRED)
find_package(Eigen3 REQUIRED)
find_package(spdlog REQUIRED)
find_package(yaml-cpp REQUIRED)
find_package(Arrow REQUIRED)
find_package(PROPOSAL REQUIRED)
find_package(BZip2 REQUIRED)
find_package(ZLIB REQUIRED)
find_package(Catch2 REQUIRED)
...

target_link_libraries (
  CORSIKA8
  INTERFACE ZLIB::ZLIB BZip2::BZip2
  Boost::filesystem CLI11::CLI11
  Eigen3::Eigen spdlog::spdlog
  yaml-cpp::yaml-cpp Parquet::parquet_static
  PROPOSAL::PROPOSAL Catch2::Catch2WithMain cnpy
  )

```

Example of SpdLog formatter

```cpp

namespace corsika {

  template <typename Type>
  auto inline format_as(Type const& arg) {
    std::ostringstream os;
    os << arg;
    return os.str();
  }

} // namespace corsika

```