Professorship of Computational Photonics
Department of Electrical and Computer Engineering
Technische Universität München

Python User Interface for the mbsolve Project

Mariem Kthiri

1 Introduction

Quantum cascade laser is a type of semiconductor emitting mid-infrared or tetrahertz portion of the electromagnetic spectrum. This frequency range allows a multitude of applications, especially in the field of spectroscopy where it enables the detection of toxic chemicals, explosives, and drugs. In this context, the mbsolve Project provides the required simulation relying on C++ programming. This engineer's practice aims at the design of a Python user interface for this project.
Chapter 2 provides the theoretical background and starting point. Chapter 3 describes the conception and realization of the project. The results are detailed in chapter 4, followed by a conclusion, discussing the outcome.

2 Theoretical Background

In this project, a high performing programming language was required for the simulation because of the large size of data and the elaborate numerical treatments. C++ fulfills these criteria not to mention that it is available everywhere and reasonably well standardized. Nevertheless, it has some drawbacks: First of all it is not interactive and implementing it for user-interfaces (especially graphic user interfaces) can be quite complex. That is why it was opted for Python for the interface which brings flexibility, interactivity and simplicity (see Tab. 1).
To sum it up, the goal is to create a common interface that enables the user to wrap different C++ libraries in Python modules and load them dynamically during the execution of the program.

C++	Python
Advantages	**Advantages**
High performance and speed	Flexibility (fast edit-build-debug cycle)
useful for intensive tasks	Interactivity
Parallelization techniques	(create, change, view objects at runtime)
Drawbacks	**Drawbacks**
Non-interactive	relatively slow
Writing user-interfaces	Limitations with memory intensive tasks
is complex	Limitations with database access

Table 1 Characterictics of C++ and Python.

3 Conception and Realization

3.1 Literature and basics

At the beginning, it was necessary to acquire some basics about quantum mechanics (density matrix, Maxwell-Bloch equations..) by reading some introductory literature [1]. A concrete example of the simulation and the numerical solving of the Maxwell-Bloch system was described in the paper of Ziolkowski [2]. QuTiP [3], a Python Toolbox for simulating the dynamics of open quantum systems, was then the starting point. It offered a variety of examples using Jupyter Notebooks: an interactive tool designed to display neatly arranged code blocks with human-friendly text which makes projects easier to manage and share, they are used in this case to describe quantum systems, such as single-atom lasing, quantum Monte Carlo trajectories, etc.

3.2 Python/C++ Interface

Creating the Python/C++ Interface was possible through different means:

- Boost.Python: Boost.Python is an open source C++ library which provides an interface for binding C++ classes and functions to Python. But it is bound to GCC, which leads to a huge dependence and its extensive use of C++ template can cause compilation problems and the consumption of a large amount of memory.

- ctpyes: ctypes is a foreign library for Python, that allows calling DLLs or shared libraries and wrapping them in Python modules. It is a convenient way to reach for a few functions within a DLL, but not to make large C++ libraries available to Python in performance-critical situations.

- SWIG (Simplified Wrapper and Interface Generator): It is a language neutral compiler that turns C/C++ declarations in scripting language interfaces. It targets Python, Tcl, Perl, MATLAB, etc.. One of its important features is that it is simple and completely automated. In fact, it creates two different files; a C/C++ source file (module_name)_wrap.c or (module_name)_wrap.cxx and a Python source file (module_name).py. The first one needs to be compiled and linked with the rest of the C/C++ library to create an extension module. The second one is the file that will be imported in Python (see Fig.1).

After installing SWIG, a Python module was created starting from an interface file and a header file written in C++. The next step was to wrap a C++ class in order to create instances of this class and modify its attributes in the Python Testprogram. Then the complete C++ library was gradually elaborated, adding each time required attributes and declarations.

3.3 Serializing input parameters

The next point to handle was the input parameters, how to extract them from an XML File and then store them with results at the end. The first option was to use exml or lxml libraries, which could parse XML files, but the resulting objects having NoType made handling them problematic. Therefore, a simple object-XML mapper for Python called dexml was applied to serialize the metadata by defining subclasses of the class dexml.Model and saving the parameters as instances.

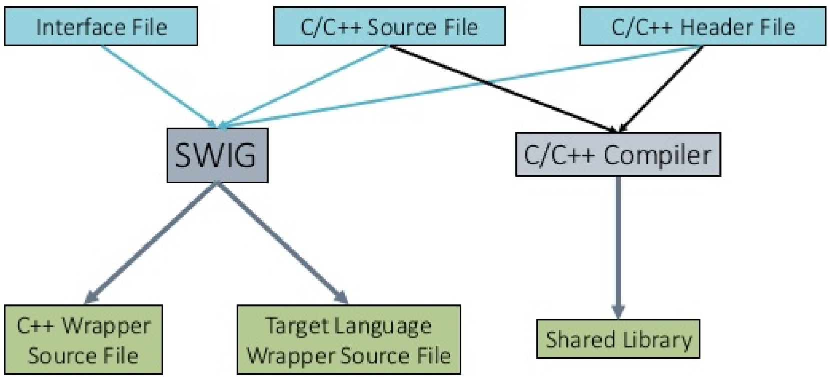

Figure 1 Functionality of SWIG
[4]

3.4 Storing results

Furthermore, it was necessary to specify the file format for the simulation results, which lead to the following options:

- XML (Extensible Markup Language): XML's biggest advantage is that it provides developers with a tool that concisely and unambiguously defines the format of data records. However, it is not suitable for large amounts of data.

- VTK (Visualization Toolkit): It is a powerful tool to visualize scalars, vectors, complex numbers, etc., but it requires a difficult setup and an understanding of the framework.

- HDF5 (Hierarchical Data Format): HDF5 is a file format that enables the storage and management of data of different types. It is suitable for high volume and complex data. In fact, an HDF5 file consists of datasets (multidimensional arrays) and groups, which contain diverse data: datasets, graphics, other groups.. Another advantage is the ability of compression and chunking, which makes the storage more flexible and efficient [5].

All that considered, HDF5 is the most suitable file format not only to store large amount of data (results) but also different types simultaneously (metadata).

3.5 Git

After settling the C++ library that contains all the necessary classes and functions, using SWIG to incorporate it in Python, choosing dexml for serialization of metadata and HDF5 for storage of results, a Git project was created to gather all the pieces, register every step of the realization and facilitate data exchange and work coordination between me and my supervisor.

3.6 Makefile/CMake

To get more experience in handling scientific projects, it was extremely efficient to understand the process of Makefiles and then CMake and apply them in my work. Both aim at building executable programs starting from many modules.

- Makefiles are a simple way to organize code compilation, they consist of a list of targets, where it is specified for each target the dependencies needed as well as the command line to apply. Besides, it only rebuilds in case of new modules added to the program.
- CMake is a cross platform build system, that automatically generates Makefiles using the CMake-Listst.txt file, that specifies the packages, libraries, source files,.. needed to build.

4 Results

Most of the work was realized on Linux and delivered the required interface consisting of different parts.

4.1 Python/C++ Interface

The user interface consists of

4.1.1 Source files

- C++ Class Record: specifies which results should be stored and in which interval.
- C++ Class Material: contains the characteristics of each material (name, electric permittivity, magnetic permeability).
- C++ Class Region: includes the specifications of each region (name, start, end, material index).
- C++ Class Device: lists the materials and regions applied in the simulation.
- C++ Class Scenario: contains the total time of the simulation, the time step and the list of the records.
- C++ Class Result: gives a vector of values corresponding to the measured result (electric field, elements of the density matrix)
- C++ Function: Depending on the device and the scenario, calculates the results.

4.1.2 CMakeLists file

SWIG can be incorporated in many build systems, in this case CMake, which can detect the SWIG executable and find the required packages and libraries for linking in order to build shared libraries. Using a single cross platform file (CMakeLists.txt) and two simple commands: cmake and make, CMake generates native build files such as makefiles, nmake files and Visual Studio projects which call SWIG and compile the generated C++ files into shared objects (.so for UNIX or .pyd for Windows).

4.1.3 Python Test program

The test program (available as a script and a notebook) allows the setup of materials, device, scenario etc., extracts metadata from an XML file, calls the C++ function that calculates the required results, displays them and stores them in addition to the metadata in an HDF5 file.
The user clones the Git repository of the project, creates a build directory, in which he runs

```
$ cmake ..
$ make
```

After bringing the test program as well as the settings XML file to the build directory, the project can be executed through

```
$ python project.py
```

Moreover, it was sought to make the project as flexible as possible, in other words compatible with different versions of Python and executable on variable operating systems.

4.2 Compatibility with Python3

SWIG is compatible with the different versions of Python (Python 2.7 and Python 3.x). It is possible to specify the wanted version in the command line while compiling with cmake by setting the option WITH_PYTHON3 as follows:

```
$ cmake -DWITH_PYTHON3=ON/OFF ..
```

CMake will find the corresponding packages and libraries and use them to compile and link.

4.3 SWIG on Windows

Another approach is available on Windows: building the extension module using a configuration file (conventionally called setup.py), it creates an extension module object using the source code files generated by SWIG, in addition to the original C++ source and compiles it into a shared object file or DLL (.pyd on Windows), which can be called in the test program.

5 Conclusion

The Python/C++ Interface is estimated to be the most suitable way to combine the efficiency and performance of C++ programming with the flexibility and interactivity of Python. SWIG was implemented to automatically create the wrappers needed to access the C++ Library from Python. Nevertheless, it was not possible in our case to make the module available system wide, because SWIG doesn't generate _init_.py files, making Python unable to recognize it as a Python module. But there are other possibilities that might overcome this restriction, such as Cython which is a Python like language for writing C/C++ extensions. In addition, further optimizations can be sought on Windows using Visual Studio.

References

[1] C. L. Tang, *Fundamentals of Quantum Mechanics* (Cambridge University Press, Cambridge, United Kingdom, 2005).

[2] D. M. C. Richard W. Ziolkowski, John M. Arnold, "Ultrafast pulse interactions with two-level atoms," Phys. Rev. A **52** (1995).

[3] J. R. Johansson, P. D. Nation, and F. Nori, "QuTiP 2: A Python framework for the dynamics of open quantum systems., Comp. Phys. Comm. 184, 1234 (2013)." .

[4] "Wrapping C ++ mit SWIG," `https://www.slideshare.net/deview/swig-c`.

[5] "HDF5 Technologies," `https://support.hdfgroup.org/about/hdf_technologies.html`.