PLext

(or *Ripping and Tearing* packages with Julia, CMake, and Python, with a little Docker for good measure)

- 🙈 Making Python faster requires external modules (C/C++/Fortan).
- 🙉 Compiling, maintaining and making these distributable can be challenging.
  - 🤦 Compiled libraries are supported by pip/wheels as an afterthought at best.
- 🙊 Compiled languages aren't as "user-friendly" and take longer to develop new features.

- Build a Julia external statevector library (libplext.so).
- Build Python/Numpy interface layer and link library (_PyPLext.*.so).
- Create Python package using above module (PyPLext).
- Build wheel for PyPI and install locally.
- Profit!

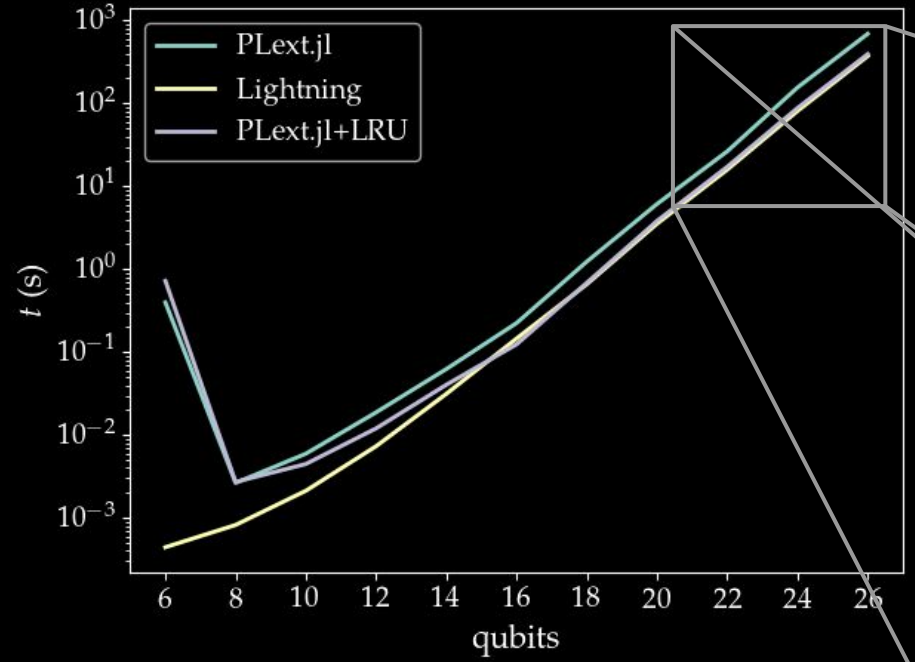
KNEE DEEP IN PACKAGE BUILDING

- External module are shared (dynamic) libraries in format of host OS (.DLL on Windows, .so/.dylib for Mac, .so for Linux)
- These files contain (almost) everything a program needs to know to use compiled functionality.
- Build binary file (.o) for each source file.
- These source files can then be packaged together (linked) to be used as a library.
- This library can then be used by another program that knows what functions are available (symbols) by linking.
- The system / compiler / flags / options / time-of-day / alignment of the moon the compiler sees can dictate whether the library can be linked (step 1) and used without error (step 2).

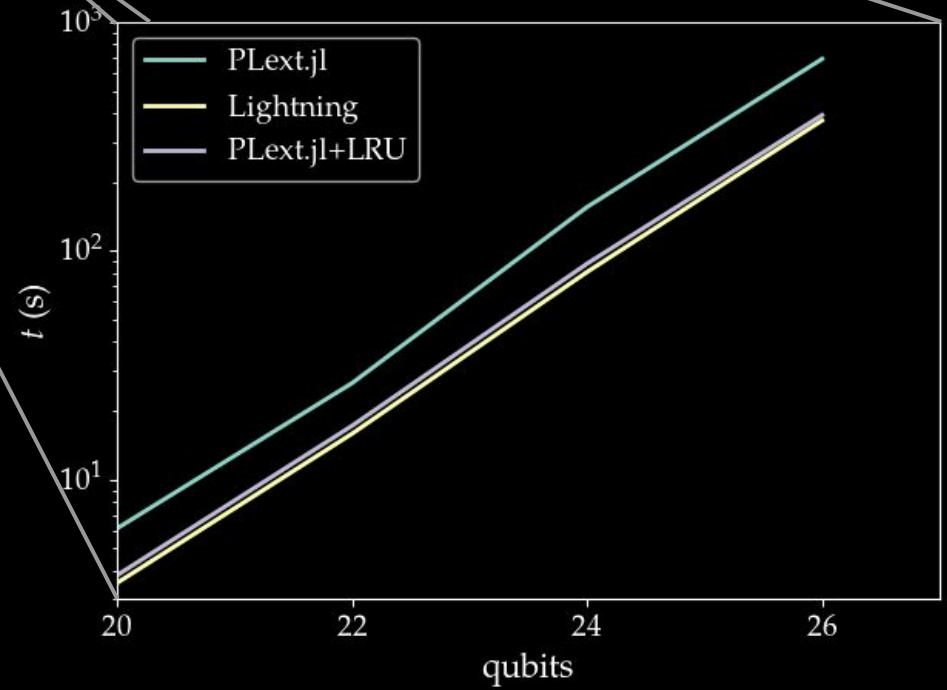
- ✓ Create PLext.jl Julia package with statevector gate kernel functions exposable to C-API.
- ✓ Set up PackageCompiler build system to compile the Julia package and all dependencies into ***libplext***.
- ✓ Create sample C++ program to build and test export works as expected.
- ✓ Create Pybind11 interface translating between Python numpy array pointer and Julia exported functions as ***_PyPLext***.
- ✓ Create Python package, ***PyPLext***, that imports compiled module and exposes functionality at Python level.

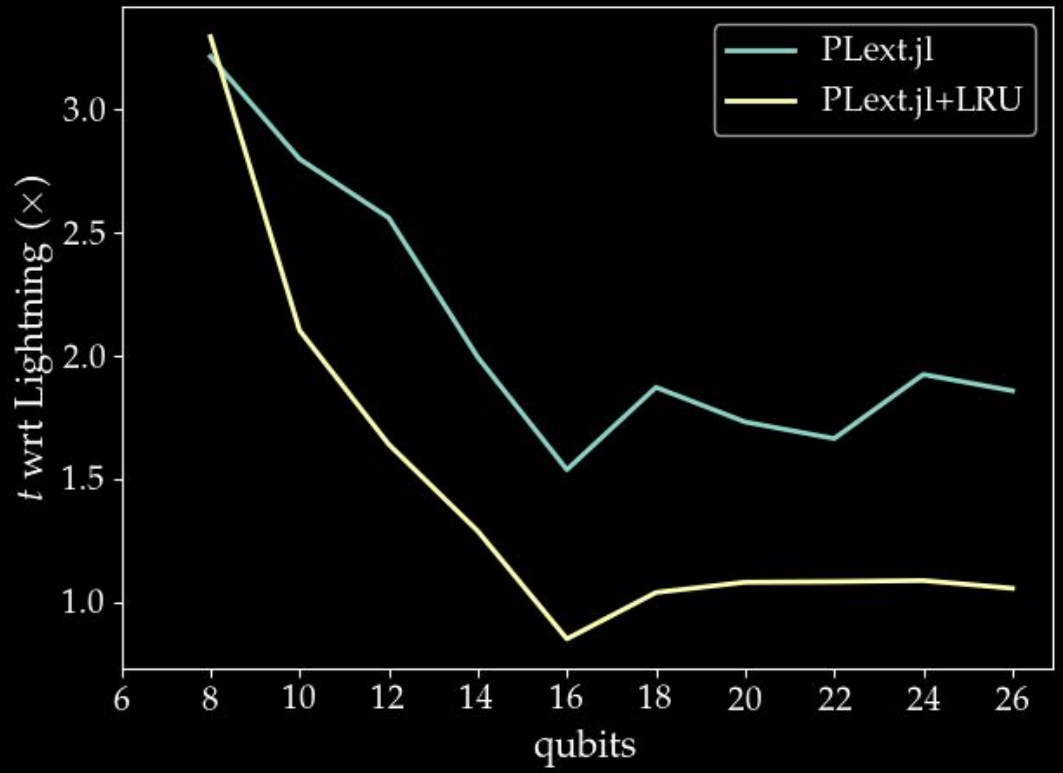
- ✓ Wheels need older C-library version to be distributable -- use *manylinux2014* Docker container.
- ✓ Add Julia to manylinux container.
- ✓ Ensure build-system is CMake & setuptools friendly to easily build and package components.
- ✓ Create wheel from all of the above!
- ✓ Patch wheel using ***auditwheel*** (includes all linked but not installed libraries).

☠ Wheel segfaults on package use.


PLext.jl
Lightning
PLext.jl+LRU
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
t (s)
6
8
10
12
14
16
18
20
22
24
26
qubits


NOT
EVERYTHING IS
TERRIBLE!


PLext.jl
Lightning
PLext.jl+LRU
10^3
10^2
10^1
t (s)
20
22
24
26
qubits



MAXIMUM PERFORMANCE

MINIMUM EFFORT

- Several packages (RAPIDS, Apache Arrow) no longer support wheel-builds, instead opting for conda only support.

- True package-management solution is needed to handle complexity of future packages: if a system isn't expressive enough, then fix or replace it.
  - Fixing pip/wheels seems like a longer term/more challenging plan.
  - Adopting conda is a much faster/easier/better solution.

- Julia, being up-and-coming, will take a larger place in scientific computing/data science ecosystem, though roadblocks still exist.

- Lightning needs an LRU cache!

https://github.com/mlxd/PLext