Pip Package for Phylanx

Project Repository : https://github.com/git-kale/phylanx_wheel

Summary of the First Evaluation

In the first evaluation, I researched different ways to create a wheel. The goal was to reuse the current build system to create wheels since this will facilitate the continuous development of the wheel as the Phylanx Project changes. I started creating a pip package for the ubuntu x86 operating system using phylanx_base: prerequisites docker. I created the wheel from the current build. I added the binary dependencies of the libraries and corrected the RPATH of those libraries. The wheel produced was installable on all Linux operating systems that supported the cp37-cp37m tag. After installation, however, the wheel didn't work as intended and failed the test cases.

Unresolved Issues during the first evaluation:

I was getting segmentation fault after running the most basic tests even after adding all the library dependencies in the wheel and overwriting RPATHs in binaries. LDD was not showing any issue in finding libraries. We suspected that the problem was with binaries themselves.

Week 5

Nikunj suggested that the problem may be with the static libraries. Therefore, I tried specifying the libraries to *extra_objects* field in setup.py.in. But the error persisted. Nikunj did a side by side comparison of Build and Wheel and found out that the libraries were replacing respective symlinks. After some searching I found out that this issue is with the wheel packaging system. The wheel packaging uses a zipfile library that lacks the support of symlink preservation. The wheel, which in turn is a zip file, hence changes those symlinks at the time of repacking or unpacking. The shared library in the package has a dependency on these libraries.
Using symlinks in the package would be preferred since keeping multiple copies of the library in a wheel leads to a huge wheel. But the wheel packaging doesn't facilitate pre/post transactions; therefore, we had to keep multiple copies of binaries in the package. I searched for possibilities to preserve symlinks after repack and unpack. But these methods would affect the CI as they are not standard. For some reason, copying the binaries didn't work; therefore, according to Prof. Hartmut's suggestion, I renamed the actual libraries, and this resolved the issue!

Week 6

After generating the wheel with proper libraries, I retested the wheel, and there got a segmentation fault. But this time, the error was a Runtime Error. I did not know how phylanx works under the hood. Prof. Hartmut suggested that the failure is due to Phylanx unable to find/load phylanx plugins. He suggested setting PHYLANX_PLUGINS_PATH environment variable to the Phylanx plugins directory. And the test ran as expected!
The test suite of Phylanx triggers in <phylanx-dir>/build; therefore, I created a test suite with the same python tests in the phylanx_wheel created a shell script to run those tests. These tests will also help in faster testing of wheels after the completion of CI.
I tested the wheel using this script. The wheel passed all the tests that the build system passes.
To test whether the wheels works as an independent entity, I installed it on a new Container, and it behaved as expected. I created a docker image using this container.

Week 7

Since I had a method that can generate wheels and the wheel as a Proof of Work. I started working on the Auditwheel approach. This approach includes creating a manylinux wheel and using auditwheel to pack the library dependencies inside the wheel. The result would be a wheel that works on almost all Linux operating systems supporting the cp37-cp37m tag. Auditwheel needs you to create the wheel package in Centos 5 container, which has a deprecated toolchain, and You need to build the package by building everything. I had a hard time building the dependencies of Phylanx, as many of them need recent toolchains. Just before building Phylanx and creating the wheel.
For checking whether auditwheel can find all the dependencies in the wheel, I moved the wheel generated earlier to the manylinux docker. I ran auditwheel check on the wheel and found out that it is only able to find extra dependencies of Phylanx [Libraries in phylanx-libs/extras]. It was unable to find HPX dependencies.
Auditwheel uses patchelf tools to get the libraries that the package needs. Phylanx project has Plugins which are necessary for the wheel. None of the libraries in the package has a dependency on these Phylanx Plugins; therefore, patchelf tools and in turn auditwheel will not be able to patch and add the Plugins to the wheel. It means that the unpacking and repacking of the wheel is necessary even after using auditwheel.
The wheel I generated does not need to link to any library outside the wheel. Hence in concept, it should work on all the Linux Operating System. I checked the wheel on Fedora 30 container that

has python3.7. At first, the wheel was throwing error. I found out that there were two libraries that I was not packaging in the wheel. I tweaked the wheel and added those libraries. The wheel worked on the Fedora system as expected!
Manylinux wheels are compatible with most of the Linux Operating systems, but building them is a tedious task and requires a lot of trial and error. Since I had a wheel that works on all Linux Operating systems, I dropped the idea to create a manylinux wheel.

Week 8

As per the suggestion of mentors, I created a development log and added it to my repo. This log will help to keep track of the history of development and errors that occurred in the project in the future. I added the CI to the repository to create a wheel in a container and update the image on Dockerhub. I created the wheel and updated the images successfully. Due to some changes in the Phylanx master branch, The system was unable to generate wheel, so I changed the CI system to work with an older commit of Phylanx. To add support for the cp38-cp38 tag, I had to replicate the same process with python3.8 instead of python3.7. I used the same phylanx_base:prerequisites docker and ran the CI, and it created the cp38-cp38 wheel successfully!
I found out that the tests in the test suite were failing, and python generated the error. I discussed it with Prof. Hartmut, and He suspected that the issue might be a result of some changes in the AST introduced with Python 3.8.

Result

A wheel package is generated that can be installed on any Linux Operating System that support cp37-cp37m tag. The wheel on testing with python tests in Phylanx passes 99/102 tests. 2 of the test cases fail due to misspelled import statement. The wheel is tested on fresh Ubuntu and fedora container. A CI is created that can produce cp37-cp37m and cp38-cp38 wheels.

Future Work

I am currently looking for How to decrease the size of the wheel. Once I will have a wheel for python3.8. I would be uploading the wheel to PyPI. I need to make some changes in the package, e.g. Adding README to the long-description field in the package. Once the wheel package passes the checks of twine, we can look into uploading it to PyPI. PyPI only supports packages up to 80MB size for default upload. Phylanx Wheel generated has a size of 358MB. Hence, I need to contact PyPI to allow the wheel on their index. There is a need to specify PHYLANX_PLUGINS_PATH manually after the

installation; I will be looking for a solution that will not require setting the environment variable manually.
The next step would be extending the CI to create wheels for Windows as well as Mac OS.
Libraries in Mac OS behave similarly to Linux. Therefore install_name_tool would replace the patchelf to correct the RPATHs, and the remaining system will work as it is.
The system will fail on Windows as there exists nothing like RPATH. In principle, the injection of some code in __init__.py to add libraries to the DLL search path should do this job. I'll search for ways to do this once I have a wheel of Mac OS.