Challenge Accepted: Python Packaging Infrastructure for the RISCV64 Ecosystem

Trevor Gamblin[1], Mark Ryan[2] and Julien Stephan[1]

[1]BayLibre Inc.
[2]Rivos Inc.

Abstract

As the RISC-V ecosystem grows with new platforms and higher performance, a key area of interest is its applicability towards scientific computing, data analysis, and machine learning. On other architectures such as ARM, these areas are already well-supported thanks to broad availability of binaries for critical packages such as NumPy, pandas, and PyTorch, but with RISC-V these are largely unavailable. The problem is further compounded by the challenge of building such packages manually, where even foundational examples such as NumPy take a prohibitively long time to build from source on commonly-available hardware, in addition to being prone to build errors and dependency chains that lead to long hours of sorting through confusing stack traces. The RISE project is tackling this problem by building, testing, deploying, and maintaining binaries for a selection of these packages. This creates a path forward for developers that wish to leverage current and upcoming RISC-V platforms for specialized research and industry applications.

Introduction

The Python "wheel", defined in [1], is the standard distribution format for Python packages. For many major Python projects whose modules consist of compiled code (for example, from C/C++), these must be built and tested on a per-architecture basis. However, while these projects distribute wheels for various architectures via online repositories such as PyPI[2], RISCV64 is currently unsupported. As a result, users who attempt to install packages on RISCV64 devices may find that the same applications which work seamlessly on other platforms do not run because one or more of the dependencies that they rely on cannot be installed.

Manual Build Challenges

Developers seeking to make use of popular Python modules used in scientific computing such as NumPy may choose to work around this situation by compiling them from source. Such projects' documentation is typically comprehensive enough that a user can initialize a build of their target project quickly. However, a more pressing problem soon presents itself: the time required to complete a build is prohibitive. The exact duration varies significantly depending on target hardware (or, alternatively, if using an emulator such as QEMU); a test run performed by the author for the NumPy project (commit cc5851e654) took approximately 20 minutes on a VisionFive V2 board running Ubuntu 24.04.1 LTS.

A 20-minute build time for the NumPy module on its own is not insurmountable, but considering that it is a core dependency for many other modules with similar build processes, the prospect that simply installing the requirements for developing data science applications will require hours of time reveals the problem inherent in this manual build approach. Furthermore, even if a developer were to undertake this challenge, there are additional complications:

1. Correctly identifying and installing the right versions of compilers and libraries required for building a specific package version is nontrivial;
2. The outputs must be distributed manually to fellow project members or other users (if they don't also perform manual builds);
3. The burden of repeating this process when new versions with additional features and bugfixes are released also falls on the individual user.

If RISCV64 platforms are to be competitive choices for such applications, this situation must be remedied. This is where the RISE Project comes in.

Developing New Packaging Infrastructre

Until the architecture can be formally supported by the Python ecosystem, an alternative solution for distributing RISCV64 wheels is necessary. By leveraging continuous integration and mirrored versions of targeted package repositories via GitLab, architecture-specific adjustments for wheel distribution tooling such as cibuildwheel[3] and manylinux[4], and emulated RISCV64 systems, binaries are built to address this deficit. These binaries are then installable with pip[5] by specifying a custom package index URL.

Current Python Ecosystem Support

As of submission, the list of packages available in the RISE Project repository[6] are as follows:

Table 1: *RISE Python Packages With RISCV64 Support*

Package	Latest Version
argon2-cffi-bindings	21.2.0
cmake	3.31.4
cffi	v1.17.1
contourpy	1.3.1
cryptography	43.0.1
httptool	0.6.4
kiwisolver	1.4.7
lintrunner	v0.12.7
lxml	lxml-5.3.0
markupsafe	3.0.2
maturin	v1.8.1
msgpack	v1.1.0
patchelf	v0.17.2.1
matplotlib	3.9.2
nh3	v0.2.18
ninja	1.11.1.3
numpy	v2.2.2
optree	v0.12.1
pandas	v2.2.2
Pillow	11.1.0
psutil	release-5.9.8
pydantic-core	v2.27.2
pyyaml	6.0.2
pyzmq	26.2.0
rpds-py	0.21.0
safetensors	v0.5.2
scipy	v1.12.0
sentencepiece	v0.2.0
tlparse	v0.3.25
tornado	6.4.2

Methodology

We used a variety of techniques to achieve the support listed in Table 1, crafting patches and customized CI pipeline branches where necessary for new versions. Build systems consisted of containerized RISCV64 environments emulated using [7] atop x86-64 AMD EPYC systems with 32 cores and 128GB of RAM, running Ubuntu 24.04 LTS. Packages were built via methods mirroring those in the upstream repositories' CI workflows, deviating mainly to limit RISE's builds to compatibility with Python versions 3.10 through 3.13. Package test suites were included before uploading binaries as a rule, with exceptions being disabled in scenarios such as when tests accessed external resources that were unavailable and/or irrelevant for RISCV64 platforms.

Adding support for Python packages on RISCV64 varied significantly in effort required. We found that there are three general levels of complexity involved:

1. **Easy**: The package requires a version update to CI scripts, followed by a rebuild.
2. **Medium**: Adjustment and patching of build backends, environment, etc. must also be performed.
3. **Hard**: Significant overhaul of existing infrastructure and/or careful analysis of test results is necessary.

Future Work

Expansion of the Python ecosystem on RISCV64 is ongoing, with support currently planned for 26[8] new packages in addition to continuing updates and builds of those in Table 1. By undertaking this challenge, we are removing the manual effort required to make use of these packages from developers, as their accessibility is now similar to that of other more established architectures. This will make adoption of RISC-V platforms in scientific computing, machine learning, educational, and hobbyist applications easier and more appealing as the hardware ecosystem evolves.

References

[1] Python.org. *Binary Distribution Format.* URL: `https://packaging.python.org/en/latest/specifications/binary-distribution-format/#binary-distribution-format`.

[2] pypi.org. *PyPI Platform Support.* URL: `https://github.com/pypi/warehouse/blob/main/warehouse/forklift/legacy.py#L158-L169`.

[3] Python Packaging Authority. *cibuildwheel.* URL: `https://github.com/pypa/cibuildwheel`.

[4] Python Packaging Authority. *manylinux.* URL: `https://github.com/pypa/manylinux`.

[5] Python Packaging Authority. *pip.* URL: `https://github.com/pypa/pip`.

[6] RISE. *RISE Project Python Repository.* URL: `https://gitlab.com/riseproject/python`.

[7] multiarch. *qemu-user-static.* URL: `https://github.com/multiarch/qemu-user-static`.

[8] RISE. *RISE wheel builder issue list.* URL: `https://gitlab.com/riseproject/python/wheel_builder/-/issues`.