PROCEEDINGS OF SCIENCE

pyQCD: A Native Lattice Simulation Package for Python

Matthew Spraggs*
University of Southampton
E-mail: ms10g12@soton.ac.uk

I demonstrate pyQCD, a native Python library providing an extensible API for single-node lattice measurements and simulations. Boost.Python is used to wrap the underlying C++ code and expose an interface to Python for the generation of propagators and configurations, both of which are returned as numpy ndarray types. The library also takes advantage of GPU technology by using CUDA where possible to accelerate Dirac operator inversions. I hope that the package will provide a set of tools for rapid prototyping and testing of lattice measurements prior to their implementation in production code.

The 32nd International Symposium on Lattice Field Theory,
23-28 June, 2014
Columbia University New York, NY

*Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

PoS(LATTICE2014)050

1. Introduction

Work in lattice field theory has produced a range of software packages used by various collaborations over the years. Due either to poor documentation or complex APIs, these bring significant challenges to those who wish to learn how lattice simulations work or experiment with new algorithms and measurements. The goal of this project (pyQCD) is to provide a Python package to satisfy these needs, whilst producing performance comparable with modern production-level code.

2. Design

2.1 Interface

I chose Python [1] as the package interface on the basis that it provides simplicity whilst requiring minimal boilerplate code. Python is an interpretted, dynamically-typed, high-level, multi-paradigm language with a focus on producing simple, expressive code. As a result of this, it is easy to learn and understand. Being an interpretted language, it is easier to debug in most cases than compiled langauges. In addition, its popularity has given rise to a large number of packages and libraries for a broad range of applications.

The major drawback of Python in the context of scientific computing is performance. The dynamically-typed nature of the language results in significant overheads, as the validity of operations must be checked at runtime. To overcome this issue, the underlying simulation code for pyQCD is written in C++ and ported to Python using Boost.Python [2], an interface to the complex Python C API. Whilst there remains some overhead when calling functions from Python, if these function calls are infrequent they will not impact significantly on the overall performance of the package.

Listing 1: Example code

```
import pyQCD
# Create the Lattice object
lattice = pyQCD.Lattice(L=4, T=8,
                        beta=5.5, action="wilson",
                        meas_spacing=10,
                        update_method="heatbath")
# Generate some configurations
for i in range(20):
    print(lattice.get_av_plaquette())
    lattice.update()
# Compute a propagator
inverter = partial(lattice.invert_wilson_dirac, mass=0.2)
source = lattice.point_source([0, 0, 0, 0])
prop = pyQCD.compute_propagator(source, inverter)
# Compute a pseudoscalar correlator
ps_correlator = pyQCD.compute_meson_corr(prop, prop, "g5", "g5")
print(ps_correlator)
```


PoS(LATTICE2014)050

The code adopts an object-orientated approach to lattice simulations, with the lattice gauge field encapsulated in a Lattice object. One can the call functions on this object to update the state of the lattice or apply actions to other objects (see Listing 1). Function output is also fully-integrated with the numpy array data type, so that these packages can be utilised, for instance when computing Dirac operator eigenvalues.

2.2 Internals

The package currently runs on a single node only. This design choice has been made to reduce the complexity of the package and avoid the complications of integrating MPI in C++ with the Python front-end. I deem this design choice reasonable at this stage, since the package is not targetting production-level calculations.

Internally, the lattice sites are arranged lexicographically, with the z-coordinate being the most rapidly varying and the t-coordinate being the least rapidly varying. This incurs the cost of poor cache performance, as hopping sites in the time-direction will likely run over cache boundaries. I recognise this to be an area where significant performance improvements could be made in future.

The Lattice class exposed to Python can be mapped directly to a corresponding class in the C++ code. This class handles allocation of the gauge links, along the implementation of the update algorithms and gauge actions. Fermionic actions are implemented in a object-oriented, modular fashion, with an arbitrary hopping matrix allowing derivative terms with arbitrary spin structure to be included in calculations with relative ease.

Extensive use is also made of the Eigen 3 linear algebra library [3]. This library is extensively optimised, with explicit vectorisation throughout. In addition, the interface is clean and simple, which will enhance the readability of the pyQCD code. I also use OpenMP [4] to parallelise certain operations.

It is also possible to compile the library to utilise any Nvidia GPUs present via the CUDA GPU framework [5]. This allows for Dirac operator inversions to be accelerated by parallelising the matrix-vector operations required in this process.

3. Further Work

There is much scope for further work on this package. The code needs to be fully profiled and optimised, particularly in the area of cache performance. Work is currently underway to produce an underlying lattice template that implements cache blocking to reduce cache misses. Part of this work also involves generalising the code to an arbitrary number of colours and dimensions. In addition, C++11 brings many useful features to the language that could be used to improve the code considerably.

Besides improving the existing code, it would be also be very useful to implement additional gauge and fermionic actions to increase the range of problems to which the package can be applied.

Work on this package is ongoing and I welcome contributions from others at the package repository, found at https://www.github.com/mspraggs/pyQCD.

PoS(LATTICE2014)050

References

[1] Python. `https://www.python.org`. Accessed: 31-10-2014.

[2] Boost.python. `http://www.boost.org/doc/libs/1_56_0/libs/python/doc/`. Accessed: 31-10-2014.

[3] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[4] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[5] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

PoS(LATTICE2014)050