The FUEL code project

James C. Osborn[*]
Argonne Leadership Computing Facility
9700 S. Cass Ave.
Argonne, IL 60439, USA
E-mail: osborn@alcf.anl.gov

We give an introduction to the FUEL project for lattice field theory code. The code being developed (called "qhmc") was initially targeted for gauge field generation for beyond standard model theories, and is now growing into a more general framework suitable for analysis too. The design is based on using the Lua scripting language as a wrapper for existing lattice field theory libraries, which provides a quick and easy way to develop new calculations. The implementation currently only supports the USQCD QOPQDP and QDP/C libraries, with support for other optimized libraries planned. We will discuss the current status of the code along with future plans for development.

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

[*]Speaker.

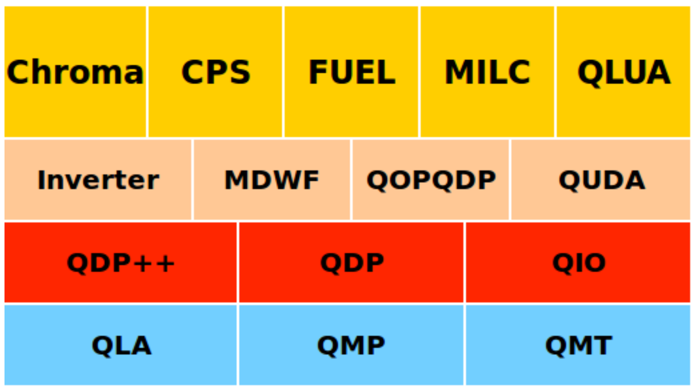

Figure 1: The USQCD lattice field theory software stack (lower 3 levels) with application codes on top.

The combination of easy to use scripting languages with optimized low level code libraries is becoming more prevalent in the high performance computing (HPC) community. This combination provides a convenient way to rapidly develop new computational workflows in a high level language while retaining the efficiency provided by the optimized machine specific libraries. Here we present a new application code project, FUEL [1], for Lattice Field Theory (LFT) calculations that provides a simple yet flexible scripting language interface on top of some of the existing optimized LFT libraries developed by the USQCD collaboration [2]. While python tends to be the most common scripting language in HPC, we have instead chosen the lightweight language Lua [3] due to its small size, ease of porting and powerful language features.

The initial goal of this project was to develop a flexible yet efficient application for the generation of gauge configurations of interest to Beyond Standard Model (BSM) physics. This required allowing for an arbitrary number of colors (N_c) in the gauge matrix along with a desire to make the rest of the code as flexible as possible (i.e. allow arbitrary numbers of dimensions, allow the addition of arbitrary field types, e.g. scalar fields, etc.). Once it started being used in production for gauge generation we began extending the analysis capabilities to allow a more complete set of analysis codes to be built using the same flexible framework.

The current FUEL application is built on top of the USQCD "C" version of the data parallel library (QDP) and its dependencies (QLA, QIO, QMP) which are described below. It also makes use of the routines (solvers, forces, etc.) available in the QOPQDP library, which is also built on QDP. These libraries provide a fairly comprehensive set of features that allows the FUEL code itself to remain relatively small. Below we will describe the features of the USQCD libraries that FUEL uses, then briefly discuss the design of FUEL itself and lastly will discuss ongoing and future development plans.

1. USQCD lattice field theory libraries

For the past 13 years, the USQCD collaboration has been developing a collection of software for lattice field theory calculations with support from the US DOE SciDAC (Scientific Discovery through Advanced Computing) initiative. A major result of this effort has been a set of libraries that provide many necessary features for lattice calculations that can be shared by all application codes [2]. The ones used by FUEL can be easily downloaded and installed using the "qinstall" script [4].

The libraries are divided into levels with the lowest being level 1, up to the full routines at level 3. Many of the library names start with "Q" for QCD though they were all designed to support more general field theories beyond QCD. The libraries being used by FUEL are

- QMP: QCD Message Passing (level 1). This abstracts the message passing interface and supports single node, MPI and Blue Gene/P and /Q SPI targets.
- QLA: QCD Linear Algebra (level 1). This provides a large set of common single-node linear algebra routines useful for LFT. The base library is compiled from C code generated by Perl scripts and has specialized $N_c = 1,2,3$ versions of all routines in addition to generic runtime selected N_c versions. There are also optimized versions of selected routines for x86 (SSE), BG/P and BG/Q (QPX) architectures.
- QIO: QCD I/O (level 2). This provides serial and parallel I/O routines for the USQCD SciDAC file format.
- QDP: QCD Data Parallel (level 2). This provides an abstraction of the fields across the lattice. It handles distribution of the data across multiple nodes using built in or user defined layouts, and can apply the large collection of linear algebra routines in QLA to all sites of the lattice, or on user defined subsets. It also handles communications for global reductions and nearest neighbor or user defined shifts.
- QOPQDP: QCD OPerations in QDP (level 3). This provides a large collection of common routines such as solvers (including multigrid for Wilson clover [5]) and force calculations.

Much of the functionality in FUEL comes from the routines available in the QOPQDP library. The main routines available are: improved gauge action, force and heatbath; staggered (plain, asqtad and HISQ) smearing, solver and force; Wilson clover solver; Wilson force (no clover term); and a Domain Wall solver. The most notable routines missing from this list are a clover force term and generalized Domain Wall variants and force terms. All of these have been planned for inclusion though the timeframe will depend on other development priorities. So far the main use for QOPQDP has been in staggered calculations (both lattice generation and analysis) and for Wilson clover analysis where the use of the existing multigrid code [5] has provided order of magnitude speedups.

2. Scripting language

There are numerous advantages to pairing a high level scripting language with optimized libraries and the idea has been making its way into HPC over the past several years. The most popular language for this use is currently Python. We initially evaluated using Python as the scripting language, but eventually decided against it due to its large code base and the related issues that arise when porting it to a new HPC architecture such as the Blue Gene/Q. Instead we have gone with the Lua [3] language with its much simpler code base which is essentially trivial to port to new architectures, and still provides a powerful and efficient language.

Lua[1] is a powerful, fast, lightweight, embeddable scripting language [6]. It is developed by a team at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro in Brazil. The whole interpreter is comprised of about 20k lines of ANSI C which makes it easy to compile on virtually any architecture. It was designed to be embedded in applications and can easily interface with C code. It is distributed under a liberal MIT license which allows code modification and redistribution. Due to its small size and permissive license we have decided to distribute the Lua code along with FUEL so that the user doesn't have to install these separately.

The FUEL executable (called "qhmc") uses a slightly modified version of the standard Lua interpreter. This allows specifying code to execute on the command line along with scripts to execute and even allows interactive use. The passing of code and scripts on the command line allows a flexible way to specify parameters to run scripts. For example

```
qhmc -e "foo=10;bar=20" setup.lua run.lua
```

will set the global variables "foo" and "bar" which will then be available to the script "setup.lua" which is executed next, and likewise any global variables set there will be available to the final script "run.lua". This allows a production workflow to divide the setup of various input parameters in arbitrary ways, and eliminates the need for developing new input file formats since the Lua scripts can already provide that functionality and more.

Another advantage of using a scripting language to drive an application is that it can greatly reduce the development time for new code. There is no need to recompile the code after each change or modify a build infrastructure when adding new files. Many scripting languages (including Lua) provide a very high level abstraction that makes creating and manipulating complex data structures easy. Lua also provides garbage collection which will automatically free objects that are no longer referenced [2]. FUEL makes use of this for all its data structures. These features make it an excellent environment for testing new algorithms or analysis methods.

3. Features and current status

The code itself was originally developed under the name "qhmc" which is still used for the code and its repository. The FUEL project name was chosen to stand for Framework for Unified Evolution of Lattices. Although the project has now extended beyond the original goal the FUEL name remains.

The original development focused on getting lattice configuration generation for staggered fermions working. This involved plugging in the staggered (asqtad and HISQ) solver and force and the improved gauge action and force routines available in QOPQDP along with a small set of utility routines for combining fields, matrix exponentiation, etc. On top of this a very flexible integrator framework was developed that allows one to specify arbitrary integration patterns (i.e. the time steps between force calculations and the time step for the corresponding momentum update). Each force term from the action can be included with its own integration pattern. The patterns for different forces are simply overlapped and as the field is integrated along in time it applies the

[1]Lua means "Moon" in Portuguese.

[2]Garbage collection only happens periodically, not necessarily as soon as an object is no longer referenced. It can be forced anytime if needed.

forces at the appropriate times according to each pattern. This provides a more general method than using a recursive integration pattern.

The staggered code has now been extended to allow for a larger range of link smearings including stout and nHYP. The corresponding gauge smearing routines are all available independent of the fermion type and can be chained together in arbitrary combinations. The corresponding force calculations can apply the chain rule to calculate the appropriate force for any user defined combination.

As the code started being used in production it became clear that having more analysis capabilities available using the same framework would be very useful. There is now a large set of operations one can perform on fields which can be used to create sources and perform contractions. Evan Weinberg at BU has contributed and tested code for staggered two-point functions including mesons, nucleons, disconnected diagrams and some gauge fixing code. This code was used for the 4+8 flavor nHYP project presented at this conference [7] where FUEL was used for the configuration generation and analysis.

FUEL also has support for the QOPQDP Wilson clover solver and the (non-clover) fermion force. Wilson configuration generation has been implemented and tested. There is currently code for meson and baryon two-point functions for both SU(3) and SU(4) gauge fields. The latter is useful for investigating composite dark matter theories [8]. This code is now being used by the Lattice Strong Dynamics collaboration in place of the previously used Chroma [9] code, and has resulted in a significant reduction in run time.

We are also using the code to investigate the use of the multigrid solver in HMC for Wilson fermions. Preliminary results of this work were presented by Meifeng Lin at this conference [10].

While the code is currently being used in several production projects, it is still a relatively new code base and is undergoing rapid development so some care is required when using it for a new project. As when starting any new project it is important to verify that the code is giving the correct results, either by comparing some cases against other existing code, or by performing consistency checks within the FUEL code. FUEL now has a regression test suite that covers several of the lattice generation and analysis routines, though it is not complete. The scripts that are currently being used for configuration generation were put together to for initial testing and development of the code and were not designed to be the final interface that the user sees, and are thus not very friendly for a new user. One of the major ongoing projects is to develop a new set of scripts with an intuitive interface that will more easily allow a user to manage a complex set of actions and algorithms and their associated parameters.

4. Development plans

As mentioned above, a major plan for the development of FUEL is to provide an easier to use interface on top of the existing code. The new interface will allow for more control of the gauge field generation procedure while trying to make the proliferation of combinations and parameters more manageable. The final design is still evolving, but has working code for pure gauge evolution. Work on adding the fermion actions is still ongoing.

A simple example of the current new interface for HMC using just a gauge action is shown here. The comments (starting with "−−") give a description of each operation.

```
L = Lattice{4,4,4,8}  -- create lattice
G = L:GaugeField{group="SU",nc=3}  -- create SU(3) gauge field
G:Load("lattice_file_name")  -- load gauge field from file
-- create gauge action
GA = Action{kind="gauge",style="plaquette",beta=6,field=G}
M = G:Momentum()  -- create momentum field in Lie group of SU(3)
MA = Action{kind="momentum",momentum=M} -- kinetic term of action
-- molecular dynamics integrator of gauge field G, with action GA
-- uses momentum M, trajectory length tau=1 with 40 time steps
I = Evolver{kind="md",style="leapfrog",action=GA,
            field=G,momentum=M,tau=1,nSteps=40}
-- Monte Carlo (Metropolis) step using sum of actions MA, GA
-- using Markov chain trial step from MD integrator I
E = Evolver{kind="mc",markov=I,actions={MA,GA},fields={G}}
-- do one HMC step
E:Run()
```

Right now most of the functionality and performance of FUEL is derived from the QDP/C library. This library has performed very well on many architectures up to and including the IBM Blue Gene/Q. However, it was not designed with threading or vectorized layouts in mind. It does have support for threading using OpenMP (inside of QLA) and there are some optimized routines for SSE and QPX vector floating point units, but the current implementation is limited in how well it can scale in the number of threads and length of the vector unit. We are currently planning to update QDP with a new version that is designed to be threaded and use vector friendly layouts, and are also investigating strategies to target CPU and GPU architectures with the same code base.

In some cases (mainly for QCD) there are already solvers and other routines optimized for specific architectures available in the various USQCD SciDAC "level 3" libraries. This includes the QUDA library for NVIDIA GPUs [11] and the QPhiX library for the Intel Xeon Phi architecture [12]. We are planning to add support for these libraries which should provide most of the performance needed on these architectures for the supported actions. For developing new actions and algorithms that are not supported by these libraries, we will rely on the QDP library, and especially its updated version when ready, to provide the flexibility and performance needed for exploring new areas of physics.

We should note that there is another application, Qlua [13], which also uses the Lua language on top of the QDP/C library and has a focus on analysis. Qlua presents a fairly complete set of Lua wrappers for the QDP field operations and uses operator overloading to allow mathematical expressions to be written. Since writing expressions generally requires the allocation of temporary fields, FUEL has avoided that approach for now in favor of function calls. We are still investigating the possibilities for adding operator overloading and an expression syntax for fields into FUEL, including the possibility of using the existing implementation in Qlua.

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, through the

Office of High Energy Physics under the SciDAC program and through the Argonne Leadership Computing Facility.

References

[1] http://jcosborn.github.io/qhmc.

[2] http://usqcd-software.github.io.

[3] http://www.lua.org.

[4] http://github.com/usqcd-software/qinstall.

[5] J. Brannick, R.C. Brower, M.A. Clark, J.C. Osborn and C. Rebbi, Phys.Rev.Lett. **100** (2008) 041601; R. Babich, J. Brannick, R.C. Brower, M.A. Clark, T.A. Manteuffel, S.F. McCormick, J.C. Osborn and C. Rebbi, Phys.Rev.Lett. **105** (2010) 201602; J.C. Osborn, R. Babich, J. Brannick, R.C. Brower, M.A. Clark, S.D. Cohen and C. Rebbi, PoS LATTICE **2010** (2010) 037.

[6] http://www.lua.org/about.html.

[7] R. Brower, A. Hasenfratz, C. Rebbi, E. Weinberg and O. Witzel, arXiv:1411.3243 [hep-lat].

[8] T. Appelquist, E. Berkowitz, R.C. Brower, M.I. Buchoff, G.T. Fleming, J. Kiskis, G.D. Kribs, M. Lin, E.T. Neil, J.C. Osborn, C. Rebbi, E. Rinaldi, D. Schaich, C. Schroeder, S. Syritsyn, G. Voronov, P. Vranas, E. Weinberg and O. Witzel, Phys. Rev. D **89** (2014) 094508 [arXiv:1402.6656 [hep-lat]].

[9] http://jeffersonlab.github.io/chroma.

[10] https://indico.bnl.gov/contributionDisplay.py?contribId=411&sessionId=10&confId=736.

[11] http://lattice.github.io/quda.

[12] http://jeffersonlab.github.io/qphix.

[13] http://usqcd-software.github.io/QLUA.html.