UKQCD software for lattice quantum chromodynamics

BY P. A. BOYLE1, R. D. KENWAY1 AND C. M. MAYNARD2,*

1SUPA, and 2EPCC, University of Edinburgh, Mayfield Road, Edinburgh EH9 3JZ, UK

Quantum chromodynamics (QCD) is the quantum field theory of the strong nuclear interaction and it explains how quarks and gluons are bound together to make more familiar objects such as the proton and neutron, which form the nuclei of atoms. UKQCD is a collaboration of eight UK universities that have come together to obtain and pool sufficient resources, both computational and manpower, to perform lattice QCD calculations. This paper explains how UKQCD uses and develops this software, how performance critical kernels for diverse architectures such as quantum chromodynamics-on-a-chip, BlueGene and XT4 are developed and employed and how UKQCD collaborates both internally and externally, with, for instance, the US SciDAC lattice QCD community.

Keywords: lattice quantum chromodynamics; quantum chromodynamics-on-a-chip; quantum field theory; CHROMA; particle physics; software

1. Introduction

Quarks and gluons are the fundamental particles that make up 99 per cent of ordinary matter, such as protons and neutrons in atomic nuclei. Quarks are bound together by the strong nuclear force, mediated by the exchange of gluons. The theory describing the strong force is called quantum chromodynamics or QCD. The strong force is actually weak when the quarks are close together but increases steadily as you try to separate them, making it impossible to isolate a single quark, a property known as ‘confinement’. This means that in experiments we detect not quarks and gluons but particles that are complicated bound states. The basic properties of the six types or flavours of quarks, such as their masses or the strengths of the interactions which turn one type of quark into another, are thus very hard to determine. The interactions changing one flavour to another are related to the small difference between matter and antimatter, called CP violation, which may help to explain why our Universe is dominated by matter (and hence why we can exist at all).

* Author for correspondence (c.maynard@ed.ac.uk).


One contribution of 16 to a Theme Issue ‘Crossing boundaries: computational science, e-Science and global e-Infrastructure I. Selected papers from the UK e-Science All Hands Meeting 2008’.
Supercomputer simulations are needed to discover whether our present theories can explain this or if there is some new physics at work. The simulations are the vital link between fundamental theories and the observed particle interactions seen in high-energy physics experiments. The calculations enable scientists to ‘look inside’ quark and gluon bound states, such as the proton and a plethora of other states known collectively as hadrons. The calculation is performed by constructing a discrete four-dimensional space–time grid (the lattice) and then solving the QCD equations of motion on this grid. Such lattice QCD simulations are the only known first-principle method for studying hadronic interactions.

2. Lattice QCD

Lattice gauge theories in general and lattice QCD in particular are the only tools we have which allow first-principles systematically improvable calculations to determine how the interactions of the fundamental quarks and gluons are translated into masses and interactions of observed particles. The computations take place using a discretized four-dimensional lattice of space–time points with spacing \( a \). The quark fields \( \psi(x) \) are four component spinors, a complex vector, and are placed on the sites of the lattice. The gluon fields \( U_{m}(x) \) are \( 3 \times 3 \) complex matrices belonging to the group SU(3). These are link variables. The equations of motion are partial differential equations, and so are replaced with finite-difference equations. These are represented by a large (proportional to the volume of the system), sparse matrix called the fermion matrix. It contains the quark–gluon coupling.

Lattice QCD calculations then proceed in two stages to compute an integral with an extremely large number of integration variables (the gluon and quark degrees of freedom across the entire lattice). First is the use of a Monte Carlo (MC) simulation to generate an ensemble of gluon field configurations with a specified probability distribution. Second is the ‘measurement’ step, the evaluation of the quantum-mechanical expectation value of an operator or operators in the background of the gauge fields. The expectation values (the averages over the gluon fields) can be converted into physical observables which can be compared with experiment. Both stages of the calculation involve solving linear systems of equations, which amounts to inverting the large, sparse fermion matrix for each configuration. This is done using Krylov subspace methods such as conjugate gradient (CG). Thus, in Philip Colella’s ‘seven dwarfs’ classification of algorithms, lattice QCD uses three: structured grids, sparse linear algebra and MC.

The fermion matrix is badly conditioned. The number of CG iterations required to invert the matrix is governed by the condition number, \( C_{N} \), which is given by

\[
C_{N} = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \times \frac{1}{m_{q}},
\]  

(2.1)

where \( \lambda_{\text{min,max}} \) are the smallest and largest eigenvalues, respectively, and \( m_{q} \) is the mass of the quark. So the number of CG iterations is inversely proportional to the quark mass. The up and down quarks are small compared with the mass of a hadron, such as a proton.\(^1\)

\(^1\)QCD has an approximate symmetry called chiral symmetry which would be exact if the up and down were massless.
The MC simulation gives a stochastic estimate of the path integral, so there is a statistical uncertainty associated. Typically, one requires $O(10^4)$ or $O(10^5)$ MC updates to generate $O(10^5)$ independent configurations. This would typically give a 1–5% statistical uncertainty for basic quantities. To control the systematic uncertainties, several simulations are required. Owing to the dependence of properties of the fermion matrix on quark mass it is not possible to simulate the physical up and down quark masses. Several simulations with different quark masses are performed, so an extrapolation can be done. For lighter quark masses, a larger volume (typically $O(10^7)$ sites) is required. At least two volumes are necessary to check for finite volume effects. Likewise, several simulations with a range of lattice spacings must be done and the results are to be extrapolated to the continuum limit of zero lattice spacing (and, at the same time, the volume must be kept large enough to avoid finite-volume errors). Typically, 20 or more simulations are required.

To summarize, lattice QCD is hard to solve because the calculation involves inverting matrices that are very large and really badly conditioned ($C_N = O(10^6)$ or more at currently accessible quark masses), and this has to be done many times ($O(10^6)$).

However, interactions in QCD are local in space and time, and so, on the lattice, only the nearest neighbours interact. This allows a parallel data decomposition with a small local volume on each processor and a simple, regular communication pattern of halo exchange. Lattice QCD calculations are dominated by matrix–vector operations and global sums in the iterative solver. There is a contention between computation and communication. The smaller the local volume, the greater the proportion of data is ‘near’ the processor. Thus each processor has less work and can access the data faster. However, the smaller the volume, the greater the number of communications of a smaller size to be performed. So the performance of lattice QCD calculations is governed by the following machine characteristics: (i) memory bandwidth and (ii) the latency and bandwidth of communication. Lattice QCD calculations are ideally suited to massively parallel computers, and this has led to the QCD community building its own machines.

3. The QCD-on-a-chip computer

Quantum chromodynamics-on-a-chip (QDCDOC; Boyle et al. 2003, 2005) uses the IBM technology library and the application-specific integrated circuit is based on the powerPC 440 embedded CPU core. It has a 64-bit FPU and can do one Fused Multiply Add per clock cycle. It has 4 MB of embedded dynamic random access memory (EDRAM) with high-bandwidth access. It also has an on-chip memory and Ethernet controller. The custom-designed elements include the following: 12 high-speed serial links with direct memory access; a pre-fetching EDRAM controller, with a bootable Joint Test Action Group (JTAG) interface. The clock runs at 400 MHz, which implies a peak speed of 0.8 Gflop s$^{-1}$ per node. The network is configured as a six-dimensional torus. It was jointly developed by Columbia University (CU), Riken Brookhaven Research Centre (RBRC), UKQCD and IBM, and it is the direct predecessor of the IBM BlueGene series of computers.

By exploiting the characteristics of QCD calculations, QDCDOC is designed to operate with a small local volume. In particular, the communications network can saturate the bandwidth for even a small packet size. This gives the network...
a low latency and allows many small messages, ideal for a small local volume. Coupled with the small, but fast memory, this allows codes to scale near perfectly out to very large numbers of processors. The UK QCDOC machines have a total of 14000 processors, and the largest individual machine is a 4096 node machine.

4. The UKQCD collaboration and SciDAC USQCD

The UKQCD collaboration has approximately 40 members from eight UK universities: (i) Cambridge, (ii) Edinburgh, (iii) Glasgow, (iv) Liverpool, (v) Oxford, (vi) Plymouth (joined in 2007), (vii) Southampton and (viii) Swansea. Prior to the QCDOC era (up to 2002), there was a consensus among members of the collaboration as to the form of the calculation. The collaboration ran one set of simulations on a central resource (Cray T3D/T3E) using collaboration-owned Fortran code, with assembler kernels for performance. Coincidently with the demise of the T3E, this consensus of opinion also ceased to exist. As a result, the collaboration now does several (three currently) different calculations. The collaboration mitigates this dilution of its central resource by each sub-group collaborating with other groups internationally. There is no Fortran compiler for QCDOC and UKQCD uses, and contributes to, two open source C++ codes, Columbia Physics System (CPS) and QDP++/CHROMA. Recently, one of the sub-projects has started working with the MILC code. Once again key kernels are written in either assembler or other architecture-specific optimizations for performance.

The US DOE-funded SciDAC programme has provided funding for the US lattice community, under the umbrella grouping of USQCD (2000, http://www.usqcd.org/). Besides funding hardware, such as QCDOC, it has also funded software development. USQCD created a unified program environment from existing codes (such as those mentioned above), while achieving high efficiency for performance and portability to new architectures. This had been achieved by the QCD applications programming interface, illustrated in figure 1.

5. Columbia Physics System

The CPS was originally developed by physicists at CU, for the QCD-signal-processor (QCDSP) machine, an ancestor of QCDOC. This code was not an American National Standards Institute (ANSI) C++ code and had many QCDSP-specific features. As such it was not readily portable, and belonged to the CU developers. UKQCD chose to use this code base as this was likely to run on QCDOC from day 1, and it had the required functionality. This was important as building your own computer already carries a big risk and this strategy reduced the total risk by having functioning code. An EPCC project ran to ANSI-fy the code. It then ran correctly, if slowly, on different platforms.

UKQCD made several key contributions to the CPS code running on QCDOC. The key kernels are in assembler, generated by BAGEL (P. Boyle 2005, http://www.ph.ed.ac.uk/~paboyle/bagel/Bagel.html), the assembler generator (see later). The UKQCD collaboration developed the rational hybrid Monte Carlo (RHMC) algorithm for QCD, and implemented this in the CPS (Clark et al. 2005; Clark 2006; Clark & Kennedy 2007). This new algorithm has many tunable parameters, and so testing for correctness and tuning for performance became a significant task.
The UKQCD collaboration ran (and continues to do so) this new algorithm in the CPS on QCDOC for two physics projects; firstly, with the RBC collaboration\textsuperscript{2} the 2 + 1 flavour Domain Wall Fermion (DWF) project. Here the ‘2’ means the up and down quark masses are degenerate, plus the ‘1’ is the strange quark. The second project was the 2 + 1 flavour AsqTad project, with collaborators in the USA, such as MILC.

Both these projects were only possible with the RHMC algorithm. Standard HMC can be used to evolve two degenerate flavours of quarks. For technical reasons, the determinant of the fermion matrix $M$ is required in the path integral, but because the quarks fields are anti-commuting Grassmann variables, the HMC algorithm cannot be constructed from the fields, but can be done from their square, i.e. two degenerate flavours. This is expressed as

$$\det(M^\dagger M).$$ (5.1)

To simulate one flavour, the square root is required

$$\det(M^\dagger M)^{1/2}. \quad (5.2)$$

The square root is approximated by a rational function, i.e.

$$r(x) = \sum_k \frac{\alpha_k}{x + \beta_k}. \quad (5.3)$$

The roots and the poles of this can be determined using a multi-shift solver.

Once this rational approximation functionality, necessary for the science, was in place, various algorithmic tricks could be deployed. For instance, the terms in the partial fraction above which contribute most to the fermion force (required

\textsuperscript{2}CU, Brookhaven National Laboratory and RBRC.
during the MC update) cost the least to compute, so the CG solving tolerance can be relaxed, and thus fewer CG iterations are needed. The tolerance is restored for the accept/reject step. This can reduce the iteration count by a factor of two.

The determinant of the fermion matrix is evaluated using pseudofermion fields,

$$\det M = \int d\phi^* \exp(-1\phi^* M^{-1} \phi) d\phi,$$

and it has been found that multiple pseudofermion fields help to reduce the noise in the simulation. This can be represented by

$$\det M = [\det(M^{1/n})]^n,$$

with \( n \) sets of pseudofermion fields. This can be combined with mass preconditioning,

$$\det(M_l^\dagger M_l)^{1/2} \cdot \det(M_s^\dagger M_s)^{3/2} = \frac{\det(M_l^\dagger M_l)}{\det(M_s^\dagger M_s)} \cdot \det(M_s^\dagger M_s)^{3/2}$$

and multiple time-scale integration. So the gluon fields are evolved on the finest time scales, the triple strange the next, and the mass preconditioned light quarks on the coarsest time scale. This allows a larger overall step size with good acceptance, with a weaker light quark mass dependence. These algorithmic optimizations have achieved a speed-up factor of 5–10. As a result of this the CPS binaries for QCDOC have been frozen since March 2006.

The CPS was written with a target architecture in mind, QCDOC. The code base runs correctly on the target hardware. This low-risk software strategy is important when building your own high-risk hardware. The adoption of the CPS allowed UKQCD to focus on its strength; in this case, a very successful algorithmic development, based on a direct collaboration with RBC. However, this is only the first stage of the calculation. Measurements on the QCDOC-generated gluon field configurations still need to be done. To do this, the fermion matrix must be inverted and the resultant quark propagators processed. UKQCD uses another set of codes to do this.

### 6. QDP++/Chroma

The second code base is an open source C++ code, used and developed by many groups around the world (Edwards & Joo 2005). The main developers are based at JLAB in Virginia, USA. UKQCD has strong historical links to this group. The code is multi-platform by design. This is achieved by a highly modular, layered design, built as a series of libraries. The QMP library is the bottom layer. It is essentially a wrapper around the message passing layer, usually MPI, but it also builds the basic four-dimensional communication pattern. The next layer, called QDP++, builds ‘lattice-valued’ physics data objects and manipulation methods. This abstraction gives this code base a major advantage. It hides the message passing layer, and thus code optimizations and architecture-specific characteristics can be implemented ‘under-the-hood’ by expert developers without having to understand the physics. Similarly, physics application developers can
implement a physics computation without having to understand the data decomposition, or underlying hardware and software fabric. QDP++ comes with several other libraries, an extensible markup language Xpath reader, LIME, an encapsulation library for lattice data types, and the QIO library, which handles the IO of lattice data objects. On top of this is the physics library itself, CHROMA, which has rich functionality.

To illustrate why this abstraction is so useful, the following example is included. The plaquette is the simplest closed path that can be formed on the lattice. The average of all the plaquettes on a configuration is the simplest calculation that can be formed. Mathematically, it is represented by the equation (6.1)

\[
P = \sum_x \sum_{\mu, \nu < \mu} \text{Re} \text{Tr}\left\{ U_\mu(x) U_\nu(x + \mu) U^\dagger_\mu(x + \nu) U^\dagger_\nu(x) \right\},
\]

where \( U_\mu(x) \) is a 3\times3 unitary matrix associated with the link starting on site \( x \) and pointing in the direction \( \mu \). This is represented graphically in figure 2. Shown below is a code snippet that calculates the average plaquette using QDP++.

```cpp
multild<LatticeColorMatrix> u(Nd);
for(int mu=1; mu < Nd; ++mu)
{
    for(int nu=0; nu < mu; ++nu)
    {
        LatticeColorMatrix tmp_0 = shift(u[nu], FORWARD, mu) * adj(shift(u[mu], FORWARD, nu));
        LatticeColorMatrix tmp_1 = tmp_0 * adj(u[nu]);
        Double tmp = sum(real(trace(u[mu]*tmp_1)));
        W_plaq += tmp;
    }
}
```

There are several things to note here. QDP++ data objects are implemented such that their mathematical combinations are correctly represented, i.e. the code knows how to multiply 3\times3 matrices together. Moreover, the application

\[
\]

Figure 2. The plaquette.
developer writes codes that look as the mathematical expression they wish to implement. Furthermore, all parallel implementations and optimizations are hidden from the application developer. This also increases portability as any architecture-specific features can be implemented, provided the interface remains the same. This is a very powerful abstraction. In particular, the separation of physics functionality from architecture and optimization functionality allows for a better future proofing of the code. The code can be amended for new architectural features without changing the physics code.

QDP++/CHROMA methodology is UKQCD’s choice for future code development, but there are some drawbacks. The CPS was chosen to generate the gluon field configurations to help offset the risk in machine building, and at the time CHROMA did not have the RHMC functionality. QDP++ makes heavy use of templates, and this can defeat some compilers. As a result, it is often only the GNU compilers that can be used. The C++ code is very advanced, which can be difficult for beginners. The measurement main program is entirely driven by structured input files. While this may be a useful feature offering great flexibility, it requires a lot of memory to register all the functions. QCDOC has a small memory, and the CHROMA main program fails to compile, it runs out of text segment. However, the CHROMA physics library does compile. This has led UKQCD to develop its own measurement code, UKhadron, on top of QDP++/CHROMA.

This uses an old-style main program that calls the relevant functions in QDP++ and CHROMA explicitly. This has some advantages, the power of QDP++ can be harnessed, and where the required functionality exists, the CHROMA physics library can be used. The code runs on QCDOC and everywhere else. A small group of developers can focus effectively on UKQCD physics requirements without becoming entangled in unwieldy general software engineering. In particular, the UKHADRON code contains most of the measurement codes for the DWF project, especially the iterative solvers that account for most of the CPU cycles consumed. An integrated analysis code is also included in UKhadron.

Besides contributing to the individual elements of both CPS and QDP++/CHROMA, UKQCD contributes performance code in the form of BAGEL. The BAGEL assembler generation library is written and maintained by Peter Boyle. It is composed of two key parts: a library to which one can program a generic reduced instruction set computing assembler kernel, and a set of programs that use the library to produce key QCD and linear algebra operations. In principle, all sorts of kernels could be written using the BAGEL library. The key architecture targets are POWERPC440 (QCDOC), BGL (BlueGene/L) and POWERIII. The assembler code thus produced is called from both CPS and CHROMA. On QCDOC and BlueGene machines, these kernels can achieve up to 50 per cent of peak performance, depending on the local lattice volume.

Shown in figure 3 is a schematic of the build structure for UKHADRON. This highly modular, layered approach to software has both advantages and disadvantages. It allows modular, independent code development, with ‘plug-in’ architecture-specific performance code. The resulting code is thus highly portable and gives high performance. However, module version and compiler version dependence can cause problems.

The fastest computers are now multi-core. For example, both BlueGene/P and the Cray XT4 are quad-core. A software development project run at EPCC for UKQCD is implementing threading in QDP++ able to use either OPENMP or...
UKQCD software

This will allow mixed-mode code, i.e. shared memory threading parallelism intra-node, and message passing inter-node. This will future-proof UKQCD software against probable highly symmetric, multiprocessing, future machines. UKQCD is also taking part in hardware development with IBM for BlueGene/Q to develop performance kernels for QCD. Finally, it is worth noting that the new PGAS languages such as Chapel, X10 and Fortress are also of potential interest for future software development to the lattice QCD community.

7. Physics

UKQCD has published many results from lattice QCD calculations from the above hardware and software, but the following calculation perhaps best highlights how effective these strategies have been. In Boyle et al. (2008) UKQCD employed
a new calculational technique called ‘twisted boundary conditions’ to access a new parameter regime, which previously had not been accessible to lattice QCD calculations. Shown in figure 4 is the comparison between lattice QCD and experimental data for the electromagnetic form factor of a pion. The diamonds are the experimental data. The triangles show the lattice data with a ‘heavy’ pion of 330 MeV. The dot-dashed line is the fit to the lattice data, and the solid line shows the resultant lattice prediction at the physical pion mass of approximately 140 MeV. This has resulted in the world’s best theoretical calculation of the charge radius of the pion. Results such as these are necessary to test the standard model of particle physics against experimental results from the large hadron collider.

The calculation was done by generating gluon field configurations in the CPS on QCDOC. The measurements were done in UKHADRON on QCDOC, University of Edinburgh’s BlueGene/L and HECToR, a Cray XT4. This demonstrates that implementing a new technique and getting high performance from diverse architectures is quite possible using UKQCD’s software.

8. Conclusions

Lattice QCD is a very complex scientific problem requiring complicated software. High performance is critical for lattice QCD codes. The UKQCD collaboration operates in a complex and changing collaborative environment, both internally and externally primarily, but not exclusively, with USQCD on several code suites that are unified into a common programming environment. This allows UKQCD to use, develop and contribute to the components that are most suited to get UKQCD’s science priorities completed while allowing for future development.

References


