

QDP++ Primer

Robert Edwards
David Richards

- Introduction
- Including QDP++
- QDP++ Basics
- Simple example

http://www.jlab.org/~dgr/qdp_tutorial

Introduction

- Implementation of Level-2 QCD-API in C++
- QDP++ codes will be **portable**, with intricacies of communication **hidden** to user.
- Benefit from optimisations on SciDAC-supported hardware.

Why C++?

- Enables object-orientated programming model, with *classes* and *templates*
- Corollary is **operator overloading** - $A * B$ has different natural meaning when A are **scalar numbers** or **matrices**
- More sophisticated I/O

Obtaining QDP++

QDP++ can be by going the to SciDAC Web Page <http://www.lqcd.org/scidac>, and looking under software.

JLab account holders:

```
cvs -d :ext:cvs.jlab.org:/group/lattice/cvsroot checkout qdp++
```

Parallel versions require QMP, but for the primer perform a straightforward scalar installation.

N.B.: you need GCC \geq 3.0.

Including QDP++

The application views QDP++ as a library , which is included in the same way as any other library, such as QMP.

At top of every file we have:

```
#include "qdp.h"
```

```
using NAMESPACE QDP;
```

Tell compiler where to look for library and include files by adding to Makefile:

```
LDFLAGS=-L${HOME}/qcd/src/qdp++/lib -lqdp
```

```
CXXFLAGS=-I${HOME}/qcd/src/qdp++/include/
```

```
hello_world: hello_world.o
```

```
$(CXX) $(CXXFLAGS) -o $@ $< ${LDFLAGS}
```

Hello, world

```
int main(int argc, char *argv[])
{
    QDP_initialize(&argc, &argv);}

    const int foo[] = {4,4,4,8};
    multi1d<int> nrow(Nd);
    nrow = foo; // Use only Nd elements
    Layout::setLattSize(nrow);
    Layout::create();

    cout << "Hello, world" << endl;
    {
        int lattice_volume;
        lattice_volume = Layout::vol();
        cout << "Volume is " << lattice_volume << endl;
    }

    QDP_finalize();
}
```

QDP++ Data Types

Scalar Classes

- **Real** - either a *float* or a *double*
- **Complex** - constructed from two **Real**
- **ColorMatrix** - A general **SU(Nc)** matrix
- **Fermion** - A complex object with **Nc** colour indices and **Ns** spinor indices

We can create arrays: `multi1d<int> nrow(Nd)`

Lattice Objects

Lattice-wide versions of the above, on which we perform **data-parallel** operations: **LatticeReal**, **LatticeColorMatrix**, **LatticeFermion**.

Can create arrays, e.g. for **gauge field**: `multi1d<LatticeColorMatrix> u(Nd)`.

QDP++ Operations

Analog of QDP functions in C implementation - operator form.

```
LatticeColorMatrix u, v, w  
w = v*u
```

corresponds to matrix multiplication applied for all sites - data-parallel.

Operator overloading The fermion expression $\psi_{\alpha i} = U_{ij} \gamma_{\alpha\beta}^1 \phi_{\beta j}$ is written

```
LatticeFermion psi  
LatticeFermion phi  
LatticeColorMatrix u
```

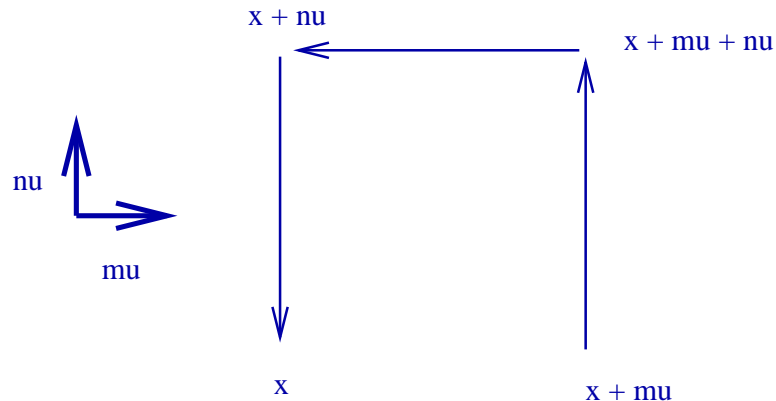
```
psi = u*Gamma(1)*phi
```

Unary operations: adj, trace

Global reductions: trace

Shifts and Communication

Simple staple $S(x) = U_\nu(x + \mu) * U_\mu(x + \nu)^\dagger * U_{\nu\mu}(x)^\dagger$ is written



```
LatticeColorMatrix S
multiid<LatticeColorMatrix> u(Nd)
int mu,nu
```

```
S = u[nu]*adj(shift(u[mu], FORWARD, nu))*adj(u[nu]);
```

shift follows convention of Fortran 90 - **FORWARD** refers to **source** relative to destination.

Subsets

Restrict operations to subset of sites, e.g. **chequerboard** $cb = 0/1$

$$cb = \text{mod}(x + y + z + t, 2).$$

Example

$$A(x) = U(x)U(x + \mu)U(x + \mu + \nu)U(x + \mu + \nu + \rho)$$

restricted to x **even**. Subset predefined in `Layout::create()`:

```
cb = 0;  
a[rb[cb]] = u*shift(u*shift(u*shift(u, FORWARD, rho), FORWARD, nu), FORWARD, mu);
```

Subsets of sources are determined by **parallel transport**.

N.B. Variables still allocated over **entire lattice** - memory is cheap!

Plaquette: `t_mesplq.cc` and `mesplq.cc`

```
...
static in tDir() (return Nd-1;) // Simple way to specify time dirn.
void MesPlq(const multiId<LatticeColorMatrix>& u,
            Double& w_plaq, Double& s_plaq, Double& t_plaq, Double& link)
{
    s_plaq = t_plaq = w_plaq = link = 0.0;

    for(int mu=1; mu < Nd; ++mu){
        for(int nu=0; nu < mu; ++nu){
            Double tmp = sum(real(trace(u[mu]*shift(u[nu],FORWARD,mu)*
                adj(shift(u[mu],FORWARD,nu))*adj(u[nu]))));

            w_plaq += tmp;
            if (mu == tDir() || nu == tDir())
                t_plaq += tmp;
            else
                s_plaq += tmp;
        }
    }
}
```

By combining the whole evaluation of the plaquette in a single expression, we avoid the use of **lattice temporaries**

Finally, we can normalise by the lattice volume:

```
w_plaq *= 2.0 / double(Layout::vol()*Nd*(Nd-1)*Nc);
```

Further examples

QDP++ has a subdirectory `qdp++/examples`:

- `dslashm_w.cc`: Wilson Dirac operator showing subsets and spin decomposition and reconstruction
- `mesons_w.cc` computes the diagonal meson correlators for Wilson fermions. It illustrates the creation of subsets associated with time-sliced sums.