

SciDAC Software Tutorial: C++

Robert Edwards

Software Coordinating Committee

February 23, 2003

Data Types

- Fields have various types (indices):

Color: $U^{ij}(x)$, Spin: $\Gamma_{\alpha\beta}$, $\psi_{\alpha}^i(x)$, $Q_{\alpha\beta}^{ij}(x)$

- Tensor product of indices forms type

	<i>Lattice</i>		<i>Color</i>		<i>Spin</i>		<i>Complexity</i>
Gauge fields :	Lattice	⊗	Matrix(Nc)	⊗	Scalar	⊗	Complex
Fermions :	Lattice	⊗	Vector(Nc)	⊗	Vector(Ns)	⊗	Complex
Scalars :	Scalar	⊗	Scalar	⊗	Scalar	⊗	Scalar
Propagators :	Lattice	⊗	Matrix(Nc)	⊗	Matrix(Ns)	⊗	Complex
Gamma :	Scalar	⊗	Scalar	⊗	Matrix(Ns)	⊗	Complex

- Some types

- Real, Complex, ColorMatrix, LatticeReal, LatticeFermion

- Can add new subtypes to support other representations (e.g., supersymmetry)

Data-parallel Operations

- *Unary and binary:*
-a; a-b; ...
- *Unary functions:*
adj(a), cos(a), sin(a), ...
- *Random numbers:*
// platform independent
random(a), gaussian(a)
- *Comparisons (booleans)*
a <= b, ...
- *Broadcasts:*
a = 0, ...
- *Reductions:*
sum(a), ...

Linear Algebra example

- Expressions short hand for index summation

$$c_{\alpha}^i(r) = Q_{\alpha\beta}^{ij}(r) b_{\beta}^j(r) + 2 d_{\alpha}^i(r) \quad \forall r$$

LatticeDiracPropagator Q;

LatticeDiracFermion b, c, d;

c = u * b + 2 * d;

Examples of data-parallel Operations

```
// Declarations
```

```
{
```

```
Real a;
```

```
LatticeReal b, c; // Memory always allocated over entire lattice
```

```
random(a); // Uniform [0,1]
```

```
gaussian(b); // Set to N(0,1)
```

```
c = 3.7 * cos(b); // Can mix constants within expressions
```

```
LatticeBoolean T = b < a; // Declare T - set to comparison at each site
```

```
} // Memory deleted upon exiting scope - destructor call
```

```
LatticeColorMatrix u, v = 1; // Broadcast - set field to 0
```

```
// Example of Dirichlet B.C. - where coord on boundary, return 0 else v
```

```
u = where(Layout::latticeCoordinate(0) == 31,0,v);
```

What is a Template?

- In C++ a **struct** declared like

```
struct foobar {  
int n; // Hold space for integer "n"  
};  
foobar A; // declare A of type foobar
```

- Can **parameterize** a class/struct/function, ...

```
template<typename T>  
struct foobar {  
T n; // make the type of "n" dependent on param T  
};  
foobar<int> A; // declare A of type foobar param'd with int
```

// CODE SKELETON

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

```
using namespace QDP;           // Bring all QDP functions into scope
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    QDP_initialize(&argc, &argv); // Initialize hardware
```

```
    multi1d<int> nrow(Nd);        // Array container of length Nd holding ints
```

```
    for(int i=0; i < Nd; ++i)
```

```
        nrow[i] = 2;            // Set the lattice size to  $2^{Nd}$ 
```

```
    Layout::setLattSize(nrow);   // Insert lattice size
```

```
    Layout::create();           // Create layout. Afterwards, QDP is usable
```

```
    // Do some wonderful and amazing things - impress your friends
```

```
    QDP_finalize();             // prepare to exit
```

```
    return 0;
```

```
}
```

```
//      Plaquette example
```

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

```
using namespace QDP;      // Bring all QDP functions into scope
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    QDP_initialize(&argc, &argv); // Initialize hardware
```

```
    multi1d<int> nrow(Nd);      // Array container of length Nd holding ints
```

```
    for(int i=0; i < Nd; ++i)
```

```
        nrow[i] = 2;          // Set the lattice size to 2^Nd
```

```
    Layout::setLattSize(nrow); // Insert lattice size
```

```
    Layout::create();         // Create layout. Afterwards, QDP is usable
```

```
    // Do some wonderful and amazing things - impress your friends
```

```
    QDP_finalize();          // prepare to exit
```

```
    return 0;
```

```
}
```