QDP++

1 Description

QDP++ is a C++ data-parallel interface for Lattice field theory applications. Many of the issues have been discussed in the corresponding C interface description which can be found at `http://www.jlab.org/ẽdwards/qcdapi/Level2API_latest.pdf`. The motivation for this development comes from the U.S. Dept. of Energy SciDAC program.

The QDP interface provides an environment somewhat similar to Fortran 90 - namely data-parallel operations (operator/infix form) which can be applied on lattice wide objects. The interface provides a level of abstraction such that high-level user code written using the API can be run unchanged on a single processor workstation or a collection of multiprocessor nodes with parallel communications. Architectural dependencies are hidden below the interface. A variety of types for the site elements are provided. To achieve good performance, overlapping communication and computation primitives are provided.

2 Supported operations

All QDP objects are of type QDPType. Supported operations are listed below. Convention: protoyypes are basically of the form:

```
QDPType  unary_function(const QDPType&)
QDPType  binary_function(const QDPType&, const QDPType&)
```

2.1 Subsets

```
Set::Make(int func(),int num) : Set construction of ordinality num subsets.
                                func maps coordinates to a coloring in [0,num)
```

2.2 Infix operators

Unary infix (e.g., "operator-"):

```
- : negation
+ : unaryplus
~ : bitwise not
! : boolean not
```

Binary infix (e.g., ”operator+”):

```
+  : addition
-  : subtraction
*  : multiplication
/  : division
%  : mod
&  : bitwise and
|  : bitwise or
^  : bitwise exclusive or
<< : left-shift
>> : right-shift
```

Comparisons (returning booleans, e.g., ”operator¡”):

```
<, <=, >, >=, ==, !=
&& : and of 2 booleans
|| : or of 2 boolean
```

Assignments (e.g., ”operator+=”):

```
=, +=, -=, *=, /=, %=, |=, &=, ^=, <<=, >>=
```

Trinary:

```
where(bool,arg1,arg2) : the C trinary "?" operator -> (bool) ? arg1 : arg2
```

2.3 Functions (standard C math lib)

Unary:

```
cos, sin, tan, acos, asin, atan, cosh, sinh, tanh,
exp, log, log10, sqrt,
ceil, floor, fabs
```

Binary:

```
ldexp, pow, fmod, atan2
```

2.4 Additional functions (specific to QDP)

Unary:

```
conj              : hermitian conjugate (adjoint)
trace             : matrix trace
real              : real part
imag              : imaginary part
colorTrace        : trace over color indices
spinTrace         : trace over spin indices
multiplyI         : multiplies argument by imag "i"
multiplyMinusI    : multiplies argument by imag "-i"
localNorm2        : on fibers computes trace(conj(source)*source)
```

Binary:

```
cmplx               : returns complex object   arg1 + i*arg2
localInnerproduct   : at each site computes trace(conj(arg1)*arg2)
```

2.5 In place functions

```
random(dest)            : uniform random numbers - all components
gaussian(dest)          : uniform random numbers - all components
zero(dest)              : zero out all elements
copymask(dest,mask,src) : copy src to dest under boolean mask
```

2.6 Global reductions

```
sum(arg1)               : sum over lattice indices returning
                          object of same fiber type
norm2(arg1)             : sum(localNorm2(arg1))
innerproduct(arg1,arg2) : sum(localInnerproduct(arg1,arg2))
sumMulti(arg1,Set)      : sum over each subset of Set returning #subset
                          objects of same fiber type
```

2.7 Accessors

Peeking and poking (accessors) into various component indices of objects.

```
peekSite(arg1,multi1d<int> coords): return site object of primitive type of arg1
peekColor(arg1,int row,int col)   : return object from color matrix elem row and col
peekColor(arg1,int row)           : return object from color vector elem row
peekSpin(arg1,int row,int col )   : return object from spin matrix elem row and col
peekSpin(arg1,int row)            : return object from spin vector elem row
```

```
pokeSite(dest,src,multi1d<int> coords): insert src at site given by coords
pokeColor(dest,src,int row,int col)   : insert src into color matrix elem row and col
pokeColor(dest,src,int row)           : insert src into color vector elem row
pokeSpin(dest,src,int row,int col )   : insert src into spin matrix elem row and col
pokeSpin(dest,src,int row)            : insert src into spin vector elem row
```

2.8 More exotic functions:

Gauge and spin related functions

- `spinProject(QDPType psi, int dir, int isign)`
  Applies spin projection $(1+isign*\gamma_\mu)$*`psi` returning a half spin vector or matrix
- `spinReconstruct(QDPType psi, int dir, int isign)`
  Applies spin reconstruction of $(1+isign*\gamma_\mu)$*`psi` returning a full spin vector or matrix
- `quarkContract13(a,b)`
  Epsilon contract 2 quark propagators and return a quark propagator. This is used for diquark constructions. Eventually, it could handle larger Nc. The numbers represent which spin index to sum over.

  The sources and targets must all be propagators but not necessarily of the same lattice type. Effectively, one can use this to construct an anti-quark from a di-quark contraction. In explicit index form, the operation `quarkContract13` does

  $$target^{k'k}_{\alpha\beta} = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'}_{\rho\alpha} * source2^{jj'}_{\rho\beta}$$

  and is (currently) only appropriate for Nc=3 (or SU(3)).
- `quarkContract14(a,b)`
  Epsilon contract 2 quark propagators and return a quark propagator.

  $$target^{k'k}_{\alpha\beta} = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'}_{\rho\alpha} * source2^{jj'}_{\beta\rho}$$
- `quarkContract23(a,b)`
  Epsilon contract 2 quark propagators and return a quark propagator.

  $$target^{k'k}_{\alpha\beta} = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'}_{\alpha\rho} * source2^{jj'}_{\rho\beta}$$
- `quarkContract24(a,b)`
  Epsilon contract 2 quark propagators and return a quark propagator.

  $$target^{k'k}_{\alpha\beta} = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'}_{\rho\alpha} * source2^{jj'}_{\beta\rho}$$

- `quarkContract12(a,b)`
  Epsilon contract 2 quark propagators and return a quark propagator.

$$target^{k'k}_{\alpha\beta} = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'}_{\rho\rho} * source2^{jj'}_{\alpha\beta}$$

- `quarkContract34(a,b)`
  Epsilon contract 2 quark propagators and return a quark propagator.

$$target^{k'k}_{\alpha\beta} = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'}_{\alpha\beta} * source2^{jj'}_{\rho\rho}$$

- `colorContract(a,b,c)`
  Epsilon contract 3 color primitives and return a primitive scalar. The sources and targets must all be of the same primitive type (a matrix or vector) but not necessarily of the same lattice type. In explicit index form, the operation colorContract does

$$target = \epsilon^{ijk}\epsilon^{i'j'k'} * source1^{ii'} * source2^{jj'} * source3^{kk'}$$

  or

$$target = \epsilon^{ijk} * source1^{i} * source2^{j} * source3^{k}$$

  and is (currently) only appropriate for Nc=3 (or SU(3)).

3 Types

Mathematically, QDP types are the product of a Nd dimensional lattice type and a type at each site (the fiber). The fiber type describes data primitives on each site. Lattice fields are defined and always allocated over the entire lattice; however, the operations can be narrowed to only a subset of sites. The primitive types at each site are represented as the (tensor) product space of, for example, a vector space over color components with a vector space over spin components and complex valued elements.

Generically objects transform under different spaces with a tensor product structure like

```
                     Color      Spin      Complexity
Gauge fields:   Product(Matrix(Nc),Scalar,    Complex)
Fermions:       Product(Vector(Nc),Vector(Ns),Complex)
Scalars:        Product(Scalar,    Scalar,    Scalar)
Propagators:    Product(Matrix(Nc),Matrix(Ns),Complex)
Gamma:          Product(Scalar,    Matrix(Ns),Complex)
```

Nd is the number of space-time dimensions
Nc is the dimension of the color vector space
Ns is the dimension of the spin vector space
NOTE: these parameters are compile time defined in `qdp++/params.h`

Gauge fields can left-multiply fermions via color matrix times color vector but is diagonal in spin space (spin scalar times spin vector). A gamma matrix can right-multiply a propagator (spin matrix times spin matrix) but is diagonal in color space (color matrix times color scalar).

Types in the QDP interface are parameterized by a variety of types including:

- *Word type*: int, float, double, bool. Basic machine types.
- *Reality type*: complex or scalar. This is where the idea of a complex number lives.
- *Primitive type*: scalar, vector, matrix, etc. This is where the concept of a gauge or spin field lives. There can be many more types here.
- *Inner grid type*: scalar or lattice. Supports vector style architectures.
- *Outer grid type*: scalar or lattice. Supports super-scalar style architectures. In combination with Inner grid can support a mixed mode like a super-scalar architecture with short length vector instructions.

There are template classes for each of the type variants listed above. The interface relies heavily on templates for composition - there is very little inheritance. The basic objects are constructed (at the users choice) by compositions like the following:

```
typedef OLattice<PScalar<ColorMatrix<Complex<float>, Nc> > > LatticeGauge
typedef OLattice<SpinVector<ColorVector<Complex<float>, Nc>, Ns> > LatticeFermion
```

The ordering of types here is suitable for a microprocessor architecture. The classes PScalar, SpinVector, ColorMatrix, ColorVector are all subtypes of a primitive type. The relative ordering of the classes is important. It is simply a user convention that spin is used as the second index (second level of type composition) and color is the third. The ordering of types can be changes. From looking at the types one can immediately decide what operations among objects makes sense. NOTE: these typdefs are defined in `qdp++/defs.h`

Operations on each level are listed below. The meaning (and validity) of an operation on the complete type (a LatticeFermion) is deduced from the intersection of these operations among each type.

3.1 Operations on subtypes

Supported operations for each type level

- ***Grid type***: *OScalar, OLattice, IScalar, ILattice*
  All operations listed in Sections 2.2–2.8

- ***Primitive type***:

  - ***PScalar***
    All operations listed in Sections 2.2–2.8

  - ***PMatrix<N>***
    Unary: `-(PMatrix)`, `+(PMatrix)`
    Binary: `-(PMatrix,PMatrix)`, `+(PMatrix,PMatrix)`, `*(PMatrix,PScalar)`, `*(PScalar,PMatrix)`, `*(PMatrix,PMatrix)`
    Comparisons: none
    Assignments: `=(PMatrix)`, `=(PScalar)`, `-=(PMatrix)`, `+=(PMatrix)`, `*=(PScalar)`
    Trinary: `where`
    C-lib funcs: none
    QDP funcs: all
    In place funcs: all
    Reductions: all

  - ***PVector<N>***
    Unary: `-(PVector)`, `+(PVector)`
    Binary: `-(PVector,PVector)`, `+(PVector,PVector)`, `*(PVector,PScalar)`, `*(PScalar,PVector)`, `*(PMatrix,PVector)`
    Comparisons: none
    Assignments: `=(PVector)`, `-=(PVector)`, `+=(PVector)`, `*=(PScalar)`
    Trinary: `where`
    C-lib funcs: none
    QDP funcs: `real`, `imag`, `multiplyI`, `multiplyMinusI`, `localNorm2`, `cmplx`, `localInnerproduct`
    In place funcs: all

Reductions: all

- ***PSpinMatrix*<*N*>**
  Inherits same operations as PMatrix
  Unary: `spinTrace`
  Binary: `*(PSpinMatrix,Gamma)`, `*(Gamma,PSpinMatrix)`
  Exotic: `peekSpin`, `pokeSpin`, `spinProjection`, `spinReconstruction`
- ***PSpinVector*<*N*>**
  Inherits same operations as PVector
  Binary: `*(Gamma,PSpinVector)`
  Exotic: `peekSpin`, `pokeSpin`, `spinProjection`, `spinReconstruction`
- ***PColorMatrix*<*N*>**
  Inherits same operations as PMatrix
  Unary: `colorTrace`
  Binary: `*(PColorMatrix,Gamma)`, `*(Gamma,PColorMatrix)`
  Exotic: `peekColor`, `pokeColor`,
- ***PColorVector*<*N*>**
  Inherits same operations as PVector
  Binary: `*(Gamma,PColorVector)`
  Exotic: `peekColor`, `pokeColor`,

- ***Reality***: *RScalar, RComplex*
  All operations listed in Sections 2.2–2.8
- ***Word***: *int, float, double, bool*
  All operations listed in Sections 2.2–2.8. Only boolean ops allowed on bool.

Note, some operations are conspicuously absent:

```
LatticeFermion foo = 1.0;     // illegal, operator=(PVector,PScalar) missing
trace(foo);                   // trace on a vector is not allowed
conj(foo);                    // illegal, there is no row vector type
```

3.2 Some known types

Some defined known types are listed below. More can easily be added or the names changed in the file `qdp++/defs.h` . These names should reflect those in the C interface.

```
Real, Integer, Double, Boolean
Complex, DComplex,
LatticeReal, LatticeInteger, LatticeComplex
LatticeFermion, LatticeColorMatrix, LatticeGauge,
LatticePropagator
```

```
ColorMatrix, ColorVector, SpinMatrix, SpinVector
LatticeColorMatrix, LatticeColorVector, LatticeSpinMatrix, LatticeSpinVector
```