QDP++: Low Level Plumbing for Chroma

Bálint Joó (bjoo@jlab.org)
Jefferson Lab, Newport News, VA

given at

HackLatt'06
NeSC, Edinburgh
March 29, 2006

Basic Plumbing: QDP++ Types

- In QDP++ we try to capture the index structure of our lattice fields:

	Lattice	Spin	Colour	Reality	BaseType
Real	Scalar	Scalar	Scalar	Real	REAL
LatticeColorMatrix	Lattice	Scalar	Matrix(Nc,Nc)	Complex	REAL
LatticePropagator	Lattice	Matrix(Ns,Ns)	Matrix(Nc,Nc)	Complex	REAL
LatticeFermionF	Lattice	Vector(Ns)	Vector(Nc)	Complex	REAL32
DComplex	Scalar	Scalar	Scalar	Complex	REAL64

- To do this we use C++ templated types:

```
typedef OScalar < PScalar    < PScalar<      RScalar <REAL>    >    > > Real;
typedef OLattice< PScalar    < PColorMatrix< RComplex<REAL>, Nc>    > > LatticeColorMatrix;
typedef OLattice< PSpinMatrix< PColorMatrix< RComplex<REAL>, Nc>, Ns> > LatticePropagator;
```

- QDP++ and Portable Expression Template Engine:
  - Provide expressions and recursion through type structure

QDP++ Expressions: The power of PETE

- Use "convenient" mathematical expressions:

```
LatticeFermion x,y,z;
Real a = Real(1);
gaussian(x);
gaussian(y);
z = a*x + y;
int mu, nu;
multi1d<LatticeColorMatrix> u(Nd);
Double tmp =  sum( real( trace(  u[mu]
                      *shift(u[nu],FORWARD,mu)
                      * adj( shift(u[mu],FORWARD,nu) )
                      *adj(u[nu])
                        )
                   )
               );
```

Wowee! No indices or for loops!

multi1d<T> = 1 dimensional array of T

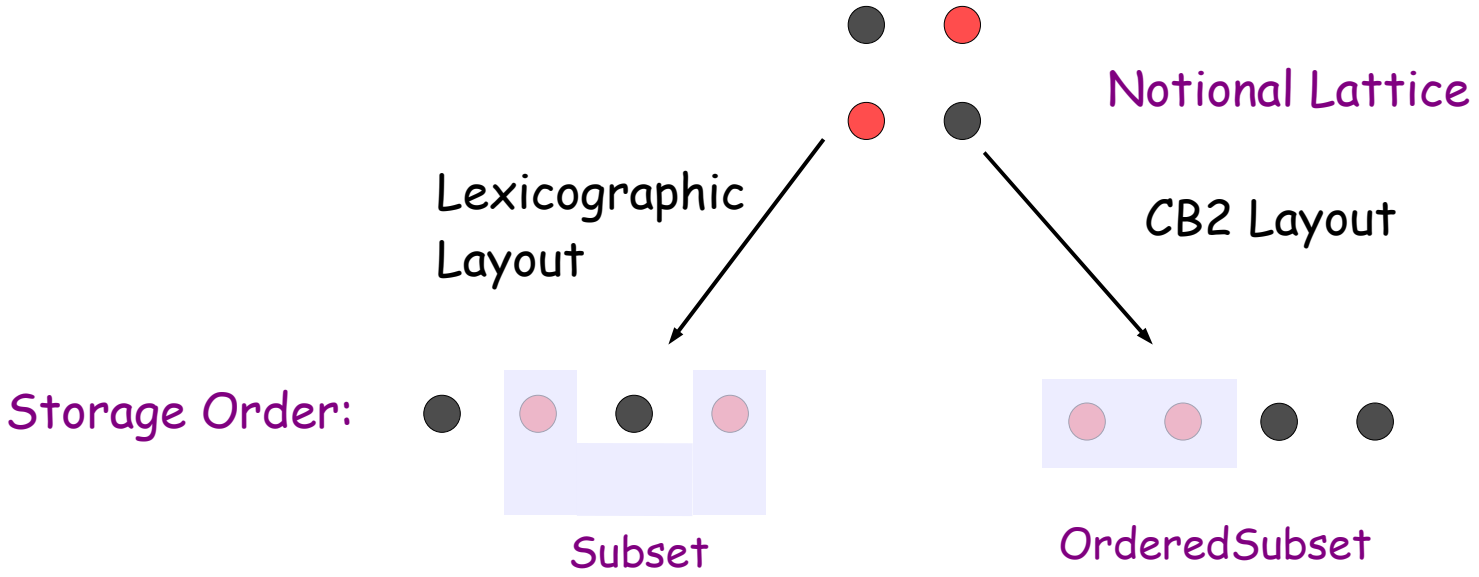
Lattice Wide Shifts FORWARD = from x+mu

Functions

but array indices are explicit

QDP++ Subsets, Ordered Subsets

- Allow for partitioning the lattice
- Can be predefined: eg rb is "red-black" colouring
- Can be contiguous (OrderedSubset) or not (Subset)
- Same subset may be contiguous in one layout and not in another



QDP++ Sets, Subsets and Layouts

- In expressions, subset index is on the target
  - bar[rb[1]] = foo (Copy foo's rb[1] subset to bar's)
- Users can define new sets
- Layout chosen at compile time with configure switch
  - Default is CB2 (2 colour checkerboard) layout
- Layout needs to be initialized on entry to QDP++

```
multi1d<int> nrow(4);
nrow[0]=nrow[1]=nrow[2]=4; nrow[3]=8;
Layout::setLattSize(nrow);
Layout::create();
```

QDP++: Reading and Writing XML

- No "static data binding" in QDP++
- Treat documents as "amorphous" (contain anything)
- Interrogate documents using XPath.
- Root of Path expression is "context node"

root node →

From root: /foo/bar/fred
from /bar: ./fred

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>
   <bar>
      <fred>6</fred>
      <jim>7 8 9</jim>
   </bar>
</foo>
```

QDP++: Reading and Writing XML

```
XMLReader r("filename");

Double y;
multi1d<Int> int_array;
multi1d<Complex> cmp_array;

try {
 read(r, "/foo/cmp_array", cmp_array);

 XMLReader new_r(r, "/foo/bar");

 read(new_r, "./int_array", int_array);
 read(new_r, "./double", y);
}
catch( const std::string& e) {
 QDPIO::cerr << "Caught exception: "
             << e <<endl;
 QDP_abort(1);
}
```

Absolute Path Read

New Context

Relative Paths

QDP++ error "stream"

try-catch exception handling construct

```
<?xml version="1.0"
      encoding="UTF-8"?>
<foo>
 <cmp_array>
  <elem>
   <re>1</re>
   <im>-2.0</im>
  </elem>
  <elem>
   <re>2</re>
   <im>3</im>
  </elem>
 </cmp_array>
 <bar>
  <int_array>2 3 4 5</int_array>
  <double>1.0e-7</double>
 </bar>
</foo>
```

Array markup of non-simple types

Array markup for simple types

Jefferson Lab
Exploring the Nature of Matter

QDP++: Reading and Writing XML

```
// Write to file
XMLFileWriter foo("./out.xml");
push(out, "rootTag");
int x=5;
Real y=Real(2.0e-7);
write(foo, "xTag", x);
write(foo, "yTag", y);
pop(out);
```

Open XML tag group

Write to a file

```
<?xml version="1.0"?>
<rootTag>
 <xTag>5</xTag>
 <yTag>2.0e-7</yTag>
</rootTag>
```

Close tag group

Write to Buffer

```
XMLBufferWriter foo_buf;
push(foo_buf, "rootTag");
int x = 5;
Real y = Real(2.0e-7);
write(foo_buf, "xTag",  x);
write(foo_buf, "yTag", y);
pop(foo_buf);
QDPIO::cout << "Buffer contains" << foo_buf.str()
            << endl;
```

write tag with content

get buffer content

QDP++ and Custom Memory Management

- Occasionally need to allocate/free memory explicitly – e.g. to provide memory to external library.
- Memory may need custom attributes (eg fast)
- Memory may need to be suitably aligned.
- May want to monitor memory usage

```
pointer=QDP::Allocator::theQDPAllocator::Instance().allocate( size,
                                                  QDP::Allocator::FAST);

QDP::Allocator::theQDPAllocator::Instance()::free(pointer);
```

Allocate memory from desired pool if possible, with alignment suitable to pool

Namespace

Get reference to allocator (see singleton pattern later)

MemoryPoolHint (attribute)

QDP++: moveToFastMemoryHint

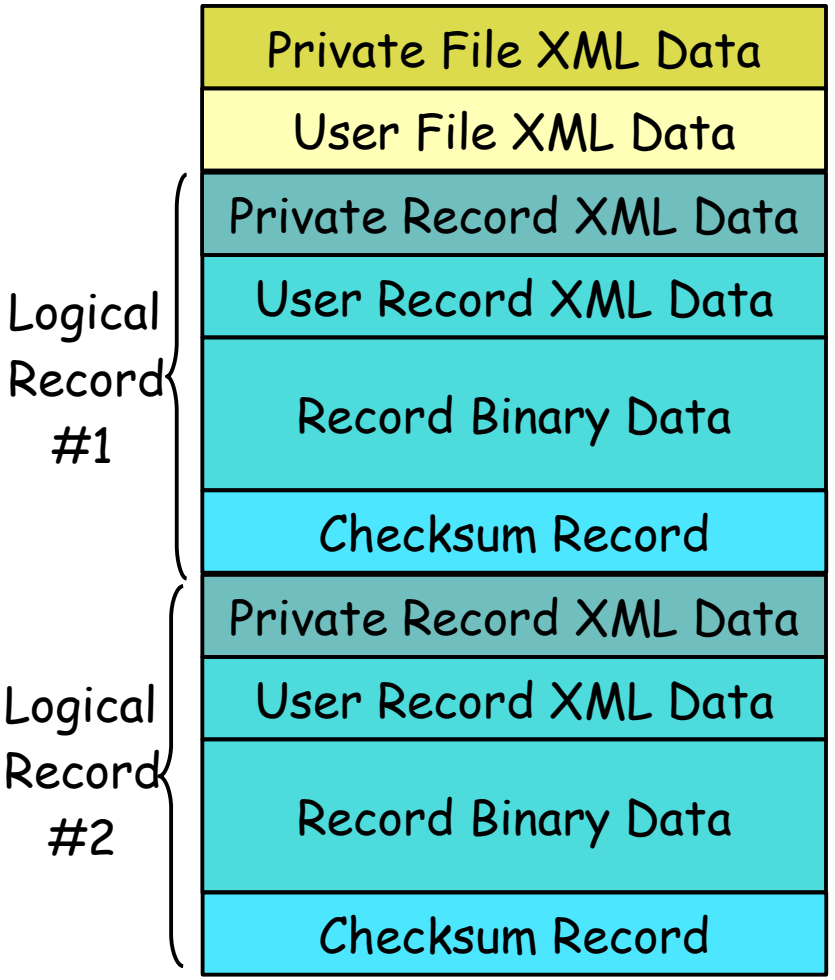
- QCDOC/SP inspired construct (copy to CRAM?)
- moves/copies data to/from fast memory (eg EDRAM)
- NOP on machines where there is no fast memory
  - It really is a hint!!!

```
LatticeFermion x;

moveToFastMemoryHint(x);

// Do some fast computation here

revertFromFastMemoryHint(x,true);
```

Accelerate x!
Do not copy contents.
Contents of x lost

Bring x back to slow memory. Copy contents

Interface to QIO



- QIO works with record oriented LIME files
- File is composed of
  - File XML records
  - Record XML records
  - Record Binary data
- SciDAC mandates checksum records
- ILDG mandates certain records

QIO Interface

- Write with QDPFileWriter
- Must supply user file and user record XML as XMLBufferWriter-s

```
LatticeFermion my_lattice_fermion;
XMLBufferWriter file_metadata;
push(file_metadata, "file_metadata");
write(file_metadata, "annotation", "File Info");
pop(file_metadata);

QDPFileWriter out(file_metadata,
                  file_name,
                  QDPIO_SINGLEFILE,
                  QDPIO_SERIAL);

XMLBufferWriter record_metadata;
push(record_metadata, "record_metadata");
write(record_metadata, "annotation", "Rec Info");
pop(record_metadata);
out.write( record_metadata, my_lattice_fermion);
out.close();
```

File XML

Record XML

QIO Write Mode Flags

- Read with QDPFileReader
- User File XML and User Record XML returned in XML Readers
- Checksum/ILDG details checked internally

```
XMLReader file_in_xml;
XMLReader record_in_xml;
QDPFileReader in(file_in_xml,
                 file_name,
                 QDPIO_SERIAL);

LatticeFermion my_lattice_fermion;
in.read(record_in_xml, my_lattice_fermion);
in.close();
```

QIO and ILDG

- The underlying QIO support layer can handle ILDG format.
- Specialization of write() function and type traits:
  - multi1d<LatticeColorMatrix> always written in ILDG format.
  - ILDG dataLFN is optional argument to the QDPFileWriter constructor.
- nersc2ildg program provided in examples/ directory
- lime_contents and lime_extract_record programs from QIO automatically built and installed

QDP++ and Efficiency

- PETE eliminates Lattice sized temporaries
- But still there are temporaries at the site level
  - This really hurts performance
- Workaround: Cliché Expressions Optimized
  - eg: a*x + y, a*x + P y, norm2(), innerProduct() etc.
  - Optimizations in C, SSE and bagel_qdp library
- Non optimized expression still slow:
  - eg: SU(3) matrix * matrix, matrix * vector, a*x+y+z
- Next round of optimization to address this issue.

QDP++ Summary

- QDP++ is an abstraction of a data parallel lattice computer with:
  - Data Parallel Expressions for Mathematics (PETE)
  - Sophisticated I/O Facilities (XML, Binary, QIO)
  - Multiple Memory Pool Memory management
- QDP++ actually has documentation (doc directory)
- Solid bedrock for Chroma and other physics apps
- Fast when using the Optimized Cliché Expressions
- Native speed to improve in SciDAC 2.