Introduction to Chroma/QDP++

David Richards, Balint Joo*

Jefferson Laboratory

HackLatt 2007
NeSC 18-20 April, 2007

* In absentia
Outline

• USQCD Software Hierarchy
  – QDP++/Chroma
• Obtaining QDP++/Chroma
• Compiling and building QDP++/Chroma
• Writing applications in chroma
  – XML, the language of chroma
  – Reading/writing XML
• Measurement routine
• Linear Operators
  – Objects
  – “Factories”: exploiting the power of Chroma
• Tutorials, further links..
SciDAC Software Structure

• See what Uncle Sam’s tax dollars are doing for you….

http://www.usqcd.org/usqcd-software
QDP++

• QDP++ is the C++ implementation of the QDP (QCD Data Parallel) interface.
• Supports lattice-wide operations:

\[
\psi_2(x) = U_\mu(x) \times \psi_1(x + \mu)
\]

\[
\begin{align*}
\text{LatticeColorMatrix} & \quad u, \ u_1, \ u_2; \\
\text{multi1d<LatticeColorMatrix>} & \quad \text{ga(Nd);} \\
\text{LatticeFermion} & \quad \text{psi1, psi2}; \\
\text{u1} & \quad = \text{shift(u, FORWARD, mu);} \\
\text{u2} & \quad = \text{shift(adj(u), BACKWARD, mu)*u;} \\
\text{psi2} & \quad = u * \text{shift(psi1, FORWARD, mu);} \\
\end{align*}
\]

• Implementation fully integrated with QMP, QIO and provides extra XML I/O features using the 3rd party LibXML2 library.
• Extensive array of examples in qdp++/examples
Chroma - I

• Application base developed in the first instance by Robert Edwards and Balint Joo, but many contributors since.
• Extensive, and evolving, application base that can both be used as a library, and as stand-alone applications.
• Runs on extensive spectrum of architectures:
  – QCDOC/BlueGene, Intel/Opteron clusters, Cray XT3/XT4,…
Obtaining Chroma and QDP++

- QMP, QDP++, and Chroma (and bundled packages) and BAGEL QDP from anonymous CVS:
  - Root
    :pserver:anonymous@cvs.jlab.org:/group/lattice/cvsroot
  - Modules: bagel_qdp, qmp, qdp++, chroma
  - Note that some versions are tagged.
- USQCD Web page:
  - [http://www.usqcd.org/usqcd/software](http://www.usqcd.org/usqcd/software)
  - Download and checkout tar-balls
Building/Compiling Chroma/QDP++ - I

- Hierarchy of builds:
  - QMP (not *scalar*)
  - QDP++
  - Chroma
- **Build directory distinct from source directory**
  - *Different architecture/precision/comms etc. with same source tree*
- For each build
  - Make build directory `build/`, cd `/build`
  - configure
  - make
  - make install
- Tricky part is configure, which needs flags specifying architecture, install directory, location of libraries etc.
  - *look in qdp++/install-scripts, chroma++/install-scripts*
- **sources in your ~/SciDAC, installed scalar build in /opt/SciDAC**
Chroma as a *library*

- Suppose chroma is installed in `/opt/SciDAC`
- Use script `chroma-config` in `/opt/SciDAC/bin`
  - `CXX=`\`chroma-config --cxx`\`
  - `CXXFLAGS=`\`chroma-config --cxxflags`\`
  - `LDFLAGS=`\`chroma-config --ldflags`\`
  - `LIBS=`\`chroma-config --libs`\`
- Compile your program (prog.cc) with:
  - `$(CXX) $(CXXFLAGS) prog.cc $(LDFLAGS) $(LIBS)`
  - NB: Ordering of flags may be important.

*Covered in tutorial*
Chroma Applications

• Measurement Application: chroma
• Gauge Generation Applications: hmc and purgaug
• Installed in same place as chroma-config
  – eg: /opt/SciDAC/bin for the tutorial
• Typical usage flags (-i, -o, -geom):
  – ./chroma -i in.xml -o out.xml -geom “Px Py Pz Pt”
  – in.xml – Input Parameter XML File
  – out.xml – Output XML Log File
  – “Px Py Pz Pt” the (possibly virtual) Processor Geometry (eg -geom “4 4 8 8” for QCDOC Rack)
Chroma - II

Measurements:
(sequential) sources, smearings propagators spectroscopy, 3pt functions, hadron structure, wilson loops, eigenvalues

Fermion Actions:
wilson, tm, clover, 4D and 5D overlap, variety of coeffs, DWF, AsqTAD

Monomials:
two flavor 4D&5D, one flavor rational 4D&5D, Hasenbusch Term (4D), LogDetEvenEven

Inverters:
CG, CGNE, BiCGStab, Multi Shift CG, SUMR, GMRESR, MINRES

Gauge Actions:
plaquette, rectangle, tree level and 1 loop LW, RG impr. plaq+rect, DBW2

Chromological Predictors:
Zero Guess, Last Solution, Linear Extrapolation, Minimum Residual

Eigensystems:
Kalkreuter-Simma Ritz

I/O Support:
NERSC, CPPACS, UKY, SciDAC and ILDG

Boundarys:
(anti)periodic, Dirichlet, twisted, Schroedinger Functional

MD Integrators:
Leapfrog, Omelyan (SW?) and Multi Time Scale versions of same

Measurement (chroma) HMC (hmc) Pure Gauge Heatbath (purgaug)
Chroma reads and speaks XML…

```xml
<?xml version="1.0" encoding="UTF-8"?>
<chroma>
    <annotation>Your annotation here</annotation>
    <Param>
        <InlineMeasurements>
            <elem>
                <Name>MAKE_SOURCE</Name>
                <Frequency>1</Frequency>
                <NamedObject>
                    <source_id>sh_source_0</source_id>
                </NamedObject>
            </elem>
            <elem>
                <Name>PROPAGATOR</Name>
                <Frequency>1</Frequency>
                <NamedObject>
                    <source_id>sh_source_0</source_id>
                    <prop_id>sh_prop_0</prop_id>
                </NamedObject>
            </elem>
        </InlineMeasurements>
        <nrow>4 4 4 8</nrow>
    </Param>
    <RNG/>
    <Cfg>
        <cfg_type>SCIDAC</cfg_type>
        <cfg_file>foo.lime</cfg_file>
    </Cfg>
</chroma>
```

Array of Measurements (Tasks)

Task (array element)

Task Name

Task specific parameters

Named Objects (communicate between tasks -- like “in memory” files)

Global Lattice Size

Input Configuration Details
XML Input File Examples

- Numerous Measurement Task Examples in
  - chroma/tests/chroma/hadron/
- Measurement Tasks for:
  - sources, smearings, propagators,
    spectroscopy, 3pt functions, eigenvalues
  - Reading and Writing Named Objects
- Also MD and HMC input files in
  - chroma/tests/t_leapfrog
  - chroma/tests/hmc
- Input file names usually contain the string “ini”
Classes: GaugeState/FermState

• In order to be useful raw gauge field states need extra info eg:
  – Boundary conditions
  – link smearing
  – eigenvectors/values

• GaugeState/FermState manages this
• Created by
  – CreateGaugeState / CreateFermState (directly)
  – GaugeAction / FermionAction (indirectly)
• Used by: LinearOperators, Gauge/Fermion Monomials
GaugeState/FermState

• Some Derivations:
  – SimpleFermState / SimpleGaugeState
    • just u and BCs
  – StoutGaugeState/ StoutFermState
  – EigenConnectState
    • u, Bcs and Fermionic Eigenvalues/Vectors

• Base Class Member Functions:
  – getLinks() - return modified links
  – deriv() - force w.r.t thin (unsmearred links)
  – getBC(), getFermBC() - get boundary conditions
FermBCs

- Interface for applying fermionic BCs
- Produced by factory
- Managed/Used by FermionAction and other GaugeBCs and FermBCs (e.g., Schroedinger Functional)

- Main members:
  - `modifyU(u)` – Apply boundaries to gauge field
  - `modifyF(psi)` – Apply boundaries to fermion field
  - `zero(F)` – Zero Force on boundary (e.g., Schroedinger functional)
**LinearOperator**

- **BaseType for matrices**
- **Templated on Fermion Type**
- **Function Object (has overloaded operator() )**

```cpp
template<typename T>
class LinearOperator {
    public:
        virtual void operator() (T& chi, const T& psi, enum PlusMinus isign) const = 0;
        virtual const OrderedSubset& subset() const = 0;
    // ... others omitted for lack of space
};
```

**PLUS apply M**

**MINUS apply M**

Know which subset to act on

**Source Vector**

**Target Vector**

**PLUS apply M**

**MINUS apply M**

Know which subset to act on
**LinearOperator**

- Created by FermionAction (factory method)
- Typical Use Pattern:

```cpp
// Raw Gauge Field
multi1d<LatticeColorMatrix> u(Nd);
typedef QDP::LatticeFermion T;
typedef QDP::multi1d<LatticeColorMatrix> P;
typedef QDP::multi1d<LatticeColorMatrix> Q;
FermionAction<T,P,Q>& S = ...;

Handle< FermState<T,P,Q> > state( S createState(u) );

Handle<LinearOperator<T> > M( S.linOp(state) );

LatticeFermion y, x;
gaussian(x);

(*M)(y, x, PLUS);
```

Create state for Fermion Kernel

Create LinearOperator (fix in links)

De-reference Handle and apply lin. op: \( y = M \times x \)
Summary (Weighing it all up)

Good Side:
- Layered – Extensive use of SciDAC and 3rd party libraries
- Very portable
- Speed through cliche-d operations and assembly
- Quality assurance: regression testing

Bad Side:
- High complexity. Can be difficult to build on some systems
- Compiler constraints
- Slow without assembly code
- Needs documentation
Chroma Development

- BlueGene – Peter Boyle
- Multicore/Threaded
- More regression testing
- Documentation
  - Anisotropic clover action
  - Temporal preconditioning
  - Stochastic Methods
Tutorial

- The tutorial is available at [http://www.jlab.org/~dgr/HackLatt07](http://www.jlab.org/~dgr/HackLatt07)
- The first tutorial is the nitty-gritty, concentrating on compiling against chroma, and computing some simple quantities, such as the pion correlator.
- The use of the chroma application, and a more extensive use of XML, is covered next – really and extension of (1) above

- The second tutorial moves onto more of the C++isms, notably the construction of linear operators above.