

Chroma tutorial Part I

Christian Hagen

Universität Regensburg

Regensburg, 20.11.2007

Outline

- 1 Chroma - A high-level LQCD library
- 2 Building Chroma
- 3 Using Chroma
- 4 Analyzing the output
- 5 Remarks
- 6 Outlook
- 7 Examples

Chroma - A high-level LQCD library

The Chroma package is a collection of LQCD applications, which

- Supports data-parallel programming constructs for lattice gauge theories especially lattice QCD
- Uses SciDAC QDP++ data-parallel programming which is written in C++
- Can generate highly optimized code for many architectures
- Contains many routines

Current contributors:

- Mainly R. Edwards and B. Joó
- but also K. Orginos, C. McNeile,

Data parallelism

- Convenient programming model
- Everything is collective
- “Shift” lattice to get at neighbors
- Global fill operations
- Operations can be limited by masks to certain lattice regions (even, odd, timeslices, ...)
- Try not to refer to an individual site (inefficient)
- Doesn't feel parallel; communication (send, receive, scatter, ...) is done behind the scenes.

Implemented modules, Part 1/6

- Gauge actions
 - Wilson gauge action
 - Lüscher-Weisz gauge action
(tree-level and 1-loop tadpole-improved)
 - RG style plaquette + rectangle gauge action
- Fermion actions
 - Staggered fermions (naïve and Asqtad)
 - Wilson fermions
 - Clover improved Wilson fermions
(Standard, SLIC, SLRC → Thomas Kaltenbrunner)
 - various approaches for Domain-Wall fermions
 - Overlap fermions (many approx. schemes in 4D and 5D and deflation)

Implemented modules, Part 2/6

- Boundary conditions
 - Periodic, antiperiodic BCs
 - Dirichlet BCs
 - Twisted BCs
- Inverters
 - Minimal Residual
 - Conjugate Gradient
 - BiCGstab
 - Conjugate Gradient with eigenvector acceleration

Most of them also as multimass version

Implemented modules, Part 3/6

- Eigenvalue/-vector measurements
 - Eigenvalue/-vector calculation à la Kalkreuter-Simma
- Gauge fixing
 - Axial gauge
 - Coulomb gauge
 - Landau gauge
 - Random gauge rotations
- Gluonic observables
 - Plaquette
 - Polyakov loops
 - Wilson loops

Implemented modules, Part 4/6

- Quark sources and sinks
 - Point
 - Shell (Gauge invariant Gaussian)
 - Derivative (Covariant derivatives and displacements)
 - Momentum wall
 - Random wall
- Link smearings
 - APE
 - HYP
 - Stout

Implemented modules, Part 5/6

- Hadronic observables
 - Meson & baryon correlators
 - Meson & baryon sequential sources
 - Meson & baryon 3pt functions
 - Meson-meson 4pt functions
 - Hybrid mesons
 - Static-light correlators
 - Static-light potentials
 - Nucleon-nucleon 4pt functions

Implemented modules, Part 6/6

- Gauge field update routines
 - Heatbath
 - HMC
 - Leapfrog and higher order integrators
 - Various one and two flavor monomials
 - Hasenbusch mass preconditioning
 - Time-scale separation
 - Various “Initial Guess” methods in 4D and 5D (last solution, linear extrapolation, MRE)
- and maybe some more

Prerequisites

Packages on your Linux system:

- libxml2 (XML for controlling Chroma and writing output)
- GNU Compiler version 4 (for the latest Chroma versions) or different compilers (Intel, Xlc++, ..., **not pgi**)
- GNU Multiple Precision library (for coefficients in RHMC)

Packages from USQCD:

- QMP (Lattice QCD Message Passing)
- QMT (Multiple Threads, might be useful for Multicore architecture and ccNuma)
- QDP++ (API used to write Chroma code, written in C++)

Getting software

- As tarballs from www.usqcd.org/software.html
- Via CVS:
 - 1 `setenv CVSROOT :pserver:anonymous@cvs.jlab.org:/group/lattice/cvsroot`
 - 2 `cvs login`
(press enter when asked for a password)
 - 3 `cvs checkout qmp`
 - 4 `cvs checkout qdp++`
 - 5 `cvs checkout qmt`
(in the following this won't be considered anymore)
 - 6 `cvs checkout chroma`
 - 7 `cvs checkout adat`
(might be needed for data analysis)

QMP - The QCD Message Passing interface

- Not necessarily needed for scalar built
- Designed by USQCD SciDAC software committee
- Simple datamodel
- Asynchronous Sends only
- Relatively easy to use/implement:
 - over MPI
 - over custom networks (QCDOC, GigE mesh)

For built via MPI:

- ```
./configure
--prefix=$QMP_INST_DIR QMP install. directory
--with-mpi
```
- ```
make
```
- ```
make install
```

# QDP++ - QCD Data Parallel interface

- API for writing Chroma code
- Can be used on its own
- Parallelization is done behind the scenes

Build instructions:

- `./configure`
  - `--prefix=$QDP++_INST_DIR` QDP++ installation directory
  - `--enable-Nd=N` # of Spacetime Dimensions, default is 4
  - `--enable-Nc=N` # of Colors, default is SU(3)
  - `--enable-Ns=N` # of Spin Components, default is 4
  - `--enable-parallel-arch=<arch>` `arch=scalar,parscalar`
  - `--enable-precision=<prec>` `prec=single,double`
  - `--enable-sse` Optimize code with Intel SSE instructions
  - `--enable-sse2` Optimize code with Intel SSE2 instructions
  - `--with-qmp=$QMP_INST_DIR` QMP installation directory
- `make`
- `make install`
- `cd docs`
- `./mkdocs.sh` to create doxygen documentation

In addition set CFLAGS and CXXFLAGS, e.g., `"-O3 -msse -msse2 -mmmx -finline-limit=50000"`

# Chroma

## Build instructions:

```
● ./configure
 --prefix=$CHROMA_INST_DIR
 --enable-sse-wilson-dslash
 --enable-cg-dwf-lowmem=<yes|no>
 --enable-opt-cfz-linop
 --with-gmp=$GMP_DIR
 --with-qmp=$QDP++_INST_DIR
```

Chroma installation directory

Build and Use the SSE2 Wilson Dslash Library

Enable Low/High Memory mode of CG-DWF inverter

Generic optimized Cont. Frac. Zolo. Linear Oper.

GMP directory

QDP++ installation directory

```
● make
● make install
● cd docs
● make
```

to create doxygen documentation

# Executables

Executables in the directory `$CHROMA_INST_DIR/bin/`

- **cfgtransf**  
transforms configurations of one format to different formats, e.g., MILC  $\rightarrow$  SCIDAC
- **chroma**  
routine for making measurements (propagators, 2pt- & 3pt-functions, smearing, ...)
- **hmc**  
routine for running HMC, but can also make measurements after update
- **purgaug**  
routine for generating quenched configurations using heatbath
- **spectrum\_s**  
spectrum code for staggered fermions
- and some testing routines

# Input/Output

Example:

```
chroma -i run.ini.xml -o run.out.xml
```

As input one used:

- run.ini.xml

xml, contains run parameters

As output one gets:

- standard output

not xml, gives information about status, performance, and timing

- run.out.xml

xml, can contain the physics, e.g., plaquettes, correlators, ...

- XMLLOG

xml, more physics but also algorithmic infos, # of CG steps, magnitude of forces, energy differences, ...

- other xml files if specified, they contain the physics

# XML files

- Executables are controlled by XML files
- Contain most simulation parameters
- XML-tags are quite intuitive  
(beta is inverse coupling, mass is a mass, ...)
- The directories in `$CHROMA_SRC/tests` contain example files for almost everything
- If not set explicitly, for some parameters default values are used
- Also the output is XML  
→ lots of unwanted output (tags)

# Output example

Example output located in plaquette.xml.100:

```
<?xml version='`1.0`'?>
<Plaquette>
 <update_no>100</update_no>
 <w_plaq>0.569168047775928</w_plaq>
 <s_plaq>0.572851951287853</s_plaq>
 <t_plaq>0.565484144264002</t_plaq>
 <plane_01_plaq>0.581934353420101</plane_01_plaq>
 <plane_02_plaq>0.572649535532255</plane_02_plaq>
 <plane_12_plaq>0.563971964911204</plane_12_plaq>
 <plane_03_plaq>0.560779522992853</plane_03_plaq>
 <plane_13_plaq>0.567773072154371</plane_13_plaq>
 <plane_23_plaq>0.567899837644782</plane_23_plaq>
 <link>-0.0108885974030071</link>
</Plaquette>
```

# print\_xpath

```
<?xml version='`1.0'`?'>
<Plaque>
 <update_no>100</update_no>
 <w_plaq>0.569168047775928</w_plaq>
 <s_plaq>0.572851951287853</s_plaq>
 <t_plaq>0.565484144264002</t_plaq>
 <plane_01_plaq>0.581934353420101</plane_01_plaq>
 <plane_02_plaq>0.572649535532255</plane_02_plaq>
 <plane_12_plaq>0.563971964911204</plane_12_plaq>
 <plane_03_plaq>0.560779522992853</plane_03_plaq>
 <plane_13_plaq>0.567773072154371</plane_13_plaq>
 <plane_23_plaq>0.567899837644782</plane_23_plaq>
 <link>-0.0108885974030071</link>
</Plaque>
```

Use the qdp++ executable print\_xpath:

```
print_xpath plaque.xml.100 /Plaque/w_plaq
```

## grep, awk, cut, sed, ...

- Good for simple things, e.g., plaquette
- Complex observables are harder to extract
- Might not work anymore when version changes

To get the plaquette use for example:

```
grep w_plaq plaquette.xml.100 |
 awk '{print $1}' | cut -c 9-22
```

# Adat

## Build instructions:

- `./configure`  
`--prefix=$ADAT_INST_DIR`    Adat install. directory
- `make`
- `make install`

Contains lots of routines to extract numbers from XML files  
(especially for spectroscopy)

Does not contain routines for everything, yet

# Remarks

## Pros:

- Many things are implemented
- Runs on basically every architecture
- Quite efficient and easy to use
- Easy to extend (probably, if you know OOP in C++)

## Cons:

- Not (well) documented, only doxygenized source code
- Maybe not most efficient on some architectures
- Might not give correct results, might need cross checks
- Many things (latest LQCD developments) are not yet implemented

# Remarks

If you consider using Chroma, don't forget to cite:

- **Always:**  
R. G. Edwards (LHPC Collaboration), B. Joó (UKQCD Collaboration),  
*"The Chroma Software System for Lattice QCD"*, arXiv:hep-lat/0409003,  
Proceedings of the 22nd International Symposium for Lattice Field Theory (Lattice2004),  
Nucl. Phys B1 40 (Proc. Suppl) p832, 2005.
- **If using SSE Optimised Dslash code (Intel P4 and other SSE compliant hardware):**  
C. McClendon,  
*"Optimized Lattice QCD Kernels for a Pentium 4 Cluster"*, Jlab preprint, JLAB-THY-01-29
- **If using output from the BAGEL assembly generator (QCDOC, IBM Power, UltraSPARC and Alpha hardware):**  
P.A. Boyle, <http://www.ph.ed.ac.uk/~paboyle/bagel/Bagel.html>, 2005

More information:

- [www.usqcd.org/software.html](http://www.usqcd.org/software.html)
- [hacklatt05-07](#)
- Balint Joó's lecture series at *INT Summer School on "Lattice QCD and its applications"* (mainly QDP++, but most examples are taken from Chroma. Thus, a nice introduction to Chroma source code)

# Outlook

In Part II:

- Source and sink smearing
- Propagator calculation
- Hadron spectroscopy
- Maybe some adat
- Hacking Chroma

# Example 1

## Hybrid Monte Carlo:

- Fermion action:  $N_f = 2$  Wilson fermions  $\kappa_{sea} = 0.11$  with even-odd preconditioning
- Gauge action: Wilson plaquette action  $\beta = 5.7$
- Lattice size:  $4^3 \times 24$
- Trajectory length:  $\tau_0 = 1.00$  divided into 50 steps  
 $\rightarrow \delta\tau = 0.02$
- MD integrator: Leapfrog
- 50 equilibration updates (warm-ups)  
10 equilibrated updates  
Measure configuration after every 2nd update

```
hmc -i hmc.prec_wilson.ini.xml -o hmc.prec_wilson.out.xml
```

# Example 1

HMC ( $N_f = 2$  Wilson fermions  $\kappa_{sea} = 0.11$ , Wilson plaquette action  $\beta = 5.7$ )

## Structure of the XML-input file hmc.prec\_wilson.ini.xml:

```
<?xml version='1.0'?>
<Params>

 <MCControl>
 Starting config, seed, number of updates, save interval, measurements
 </MCControl>

 <HMCTrj>
 Monomials, integrators, lattice size
 </HMCTrj>

</Params>
```

# Example 1, cont.

HMC ( $N_f = 2$  Wilson fermions  $\kappa_{sea} = 0.11$ , Wilson plaquette action  $\beta = 5.7$ )

## MCControl block:

```
<Cfg>
 <cfg_type>DISORDERED</cfg_type>
 <cfg_file>DUMMY</cfg_file>
</Cfg>
<RNG>
 <Seed>
 <elem>11</elem>
 <elem>0 </elem>
 <elem>0 </elem>
 <elem>0 </elem>
 </Seed>
</RNG>
<StartUpdateNum>0</StartUpdateNum>
<NWarmUpUpdates>50</NWarmUpUpdates>
<NProductionUpdates>1000</NProductionUpdates>
<NUpdatesThisRun>60</NUpdatesThisRun>
<SaveInterval>2</SaveInterval>
<SavePrefix>configname</SavePrefix>
<SaveVolfmt>SINGLEFILE</SaveVolfmt>
<ReproCheckP>>false</ReproCheckP>
<InlineMeasurements>
Here one can place measurements, e.g., correlator calculations
</InlineMeasurements>
```

# Example 1, cont.

HMC ( $N_f = 2$  Wilson fermions  $\kappa_{sea} = 0.11$ , Wilson plaquette action  $\beta = 5.7$ )

## HMCTrj block:

```
<Monomials>
Fermion and gauge monomials
</Monomials>

<Hamiltonian>
 <monomial_ids>
 <elem>wilson_two_flav</elem>
 <elem>wilson_gauge</elem>
 </monomial_ids>
</Hamiltonian>

<MDIntegrator>
 <tau0>1</tau0>
 <Integrator>
 <Name>LCM_STS_LEAPFROG</Name>
 <n_steps>50</n_steps>
 <monomial_ids>
 <elem>wilson_two_flav</elem>
 <elem>wilson_gauge</elem>
 </monomial_ids>
 </Integrator>
</MDIntegrator>
<nrow>4 4 4 24</nrow>
```

# Example 1, cont.

HMC ( $N_f = 2$  Wilson fermions  $\kappa_{sea} = 0.11$ , Wilson plaquette action  $\beta = 5.7$ )

## Fermion monomials:

```
<elem>
 <Name>TWO_FLAVOR_EOPREC_CONSTDET_FERM_MONOMIAL</Name>
 <InvertParam>
 <invType>CG.INVERTER</invType>
 <RsdCG>1.0e-7</RsdCG>
 <MaxCG>1000</MaxCG>
 </InvertParam>
 <FermionAction>
 <FermAct>WILSON</FermAct>
 <Kappa>0.11</Kappa>
 <FermionBC>
 <FermBC>SIMPLE_FERMBC</FermBC>
 <boundary>1 1 1 -1</boundary>
 </FermionBC>
 </FermionAction>
 <ChronologicalPredictor>
 <Name>LAST_SOLUTION_4D_PREDICTOR</Name>
 </ChronologicalPredictor>
 <NamedObject>
 <monomial_id>wilson_two_flav</monomial_id>
 </NamedObject>
</elem>
```

# Example 1, cont.

HMC ( $N_f = 2$  Wilson fermions  $\kappa_{sea} = 0.11$ , Wilson plaquette action  $\beta = 5.7$ )

## Gauge monomials:

```
<elem>
 <Name>GAUGE_MONOMIAL</Name>
 <GaugeAction>
 <Name>WILSON_GAUGEACT</Name>
 <beta>5.7</beta>
 <GaugeBC>
 <Name>PERIODIC_GAUGEBC</Name>
 </GaugeBC>
 </GaugeAction>
 <NamedObject>
 <monomial_id>wilson_gauge</monomial_id>
 </NamedObject>
</elem>
```

## Example 2

Eigenvalue/-vector computation:

- Unpreconditioned Wilson operator with  $am = -1.4$
- Using Kalkreuter-Simma algorithm (minimizes Ritz functional)
- Relative residuum =  $10^{-6}$ , absolute residuum =  $10^{-7}$
- 50 Eigenvalues/-vectors
- Write eigenvalues into separate xml file  
`./eigenvals_unprec_wilson.dat.xml`
- Write results to file `./eigen`

```
chroma -i eigen.unprec.wilson.ini.xml -o eigen.unprec.wilson.out.xml
```

## Example 2

### CHROMA (Eigenvalues and eigenvectors of the hermitian Wilson operator)

#### Structure of the XML-input file:

```
<?xml version='`1.0``'?>
<chroma>
<Param>
 <InlineMeasurements>

 </InlineMeasurements>
 <nrow>4 4 4 24</nrow>
</Param>
<RNG>
 <Seed>
 <elem>11</elem>
 <elem>11</elem>
 <elem>11</elem>
 <elem>0</elem>
 </Seed>
</RNG>
<Cfg>
 <cfg_type>WEAK.FIELD</cfg_type>
 <cfg_file>DUMMY</cfg_file>
</Cfg>
</chroma>
```

## Example 2, cont.

### Inline Measurements, RITZ\_KS\_HERM\_WILSON:

```
<elem>
 <Name>RITZ_KS_HERM_WILSON</Name>
 <Frequency>1</Frequency>
 <Param>
 <version>1</version>
 <FermionAction>
 <FermAct>UNPRECONDITIONED_WILSON</FermAct>
 <Mass>-1.4</Mass>
 <boundary>1 1 1 -1</boundary>
 </FermionAction>
 <RitzParams>
 <Neig>50</Neig><Ndummy>4</Ndummy>
 <RsdR>1.0e-7</RsdR><RsdA>1.0e-8</RsdA>
 <RsdZero>5.0e-6</RsdZero>
 <ProjApsiP>false</ProjApsiP>
 <GammaFactor>0.1</GammaFactor>
 <MaxKS>200</MaxKS>
 <MaxCG>10000</MaxCG>
 <MinKSIter>5</MinKSIter>
 <MaxKSIter>200</MaxKSIter>
 <Nrenorm>10</Nrenorm>
 </RitzParams>
 </Param>
 <NamedObject>
 <gauge_id>default_gauge_field</gauge_id>
 <eigen_id>eigen_info_0</eigen_id>
 </NamedObject>
 <xml.file>./eigenvals_unprec_wilson.dat.xml</xml.file>
</elem>
```

## Example 2, cont.

### Inline Measurements, QIO\_WRITE\_NAMED\_OBJECT:

```
<elem>
 <Name>QIO_WRITE_NAMED_OBJECT</Name>
 <Frequency>1</Frequency>
 <NamedObject>
 <object_id>eigen_info_0</object_id>
 <object_type>EigenInfo</object_type>
 </NamedObject>
 <File>
 <file_name>./eigen</file_name>
 <file_volfmt>SINGLEFILE</file_volfmt>
 </File>
</elem>
```