BRIDGE++: AN OBJECT-ORIENTED C++ CODE FOR LATTICE SIMULATIONS

UEDA Satoru (KEK)
Bridge++ Project

Lattice 2013 @ Mainz, Germany August 2, 2013

WHAT IS BRIDGE++ PROJECT

We are developing a lattice QCD code set Bridge++

- Project site:
  - http://bridge.kek.jp/Lattice-code/
- Core members: S.AOKI(Kyoto Univ.), T.AOYAMA(Nagoya Univ.), K.KANAYA, Y.NAMEKAWA, H.NEMURA, Y.TANIGUCHI, N.UKITA(Tsukuba Univ.), H.MATSUFURU, S.UEDA(KEK), S.MOTOKI(Aizu Univ.)
- Supported by :
  - Grant-in-Aid for Scientific Research on Innovative Areas (2008-2012) http://bridge.kek.jp/
  - Joint Institute for Computational Fundamental Science (2011-2015) http://www.jicfus.jp/
  - HPCI Strategic Program Field 5 (2011-2015) http://www.jicfus.jp/field5/

HISTORY OF 'BRIDGE++'

- 2009 project started
  - Named from the Grant "Research on the Emergence of Hierarchical Structure of Matter by Bridging Particle, Nuclear and Astrophysics in Computational Science"
  - 79 meetings have been held every 1-2weeks
  - Advices given by experts in computer science and applied mathematics
- 2012 July: ver.1.0 released
- 2013 23rd July: ver.1.1 released

OUTLINE

- Introduction
- Implementation sample:
  - Solver & fermion operator
  - HMC integrator
- Documentation
- Code tuning (In progress)
- Summary

LATTICE QCD

Recent lattice simulations require:

- Various physical models (beyond SM etc.)
- Variety of architectures (massively parallel multi-level processer, GPGPU etc.)
- Efficient numerical algorithms

- Code development becomes more involved.
- Difficult to start for students and other field researcher.

DEVELOPMENT POLICY

- Readability: easy to read and use
- Portability: from laptop PC to HPC.
- Extensibility: easy to test new ideas
- High-performance: enough for productive run

- Programming language: C++
  - Object oriented
  - Design patterns
- Covers wide range of architectures
  - MPI, OpneMP/pthread, OpenCL for arithmetic accelerators.
- Rich documents, Lots of test modules.

STATUS OF IMPLEMENTATION: HMC

- Public released:
  - Action: Plaquette/Rectangle gauge, Wilson/clover fermion
  - Smearing APE/HYP with stout projection
  - Multi-time step HMC/RHMC
- Now being confirmed:
  - Staggered, Twisted mass, Domain-wall, Overlap, Isochemical Wilson/clover
  - $N_C \neq 3$
- Being developed:
  - Adjoint fermion

STATUS OF IMPLEMENTATION: OBSERVABLE, HARDWARE ETC

- Public released:
  - Hadron spectrum, Wilson loop, Gradient flow
  - Schrödinger functional
  - CG, BiCGStab, GMRES(m) etc.
  - ILDG configuration format
  - YAML parameter file
- Now being confirmed:
  - Quark number Susceptibility
- Being developed:
  - OpenMP/pthread, OpenCL, CUDA

TEST MODULE

About 40 test modules are provided.

- Implementation samples, how to use the classes
- Verification tool

Interactive test manager

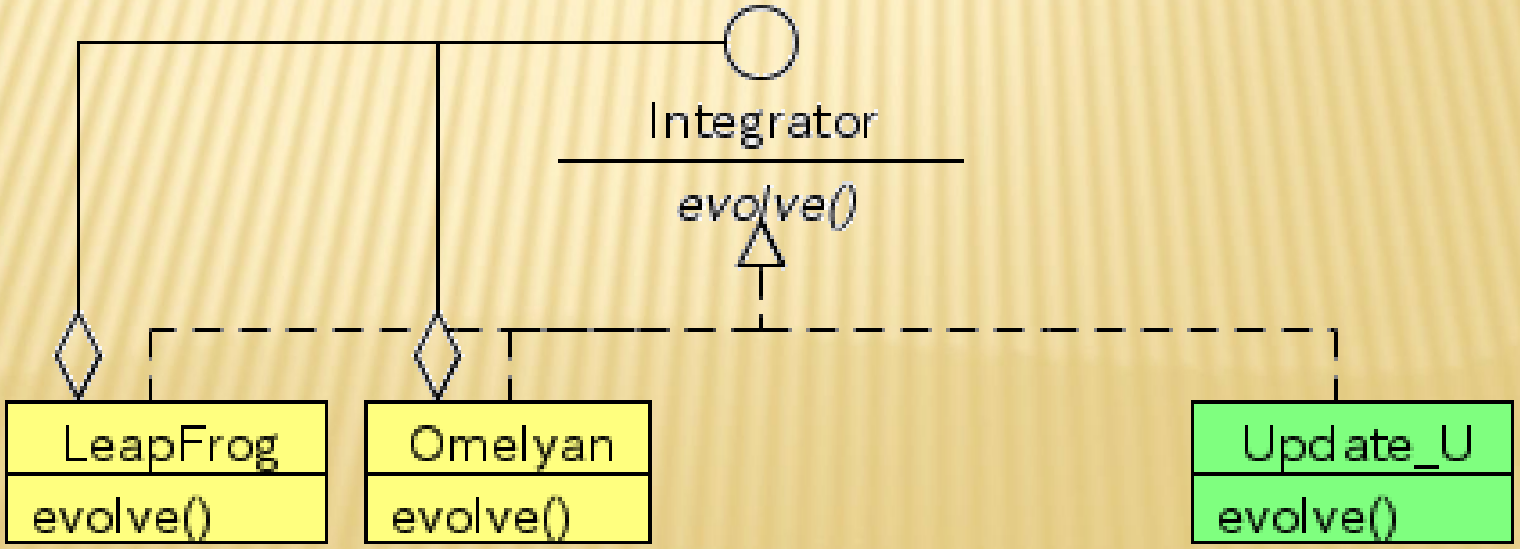
```
------------------------------------------------
Please select test category name
1 : Eigensolver
2 : Gauge
3 : GradientFlow
4 : HMC
5 : IO
6 : RandomNumbers
7 : Rational
8 : SF_fAfP
9 : ShiftSolver
10 : Spectrum
11 : WilsonLoop
a : Test All
p : Setup test check precision (current precision: 12)
q : Quit
choice> 4
1 : Clover
2 : Clover_SF
3 : Quenched
4 : Wilson
a : Test All
p : Setup test check precision (current precision: 12)
u : Go back
q : Quit
choice> 1
1 : Leapfrog_Nf2
2 : Leapfrog_Nf2_eo
3 : RHMC_Nf2p1
4 : RHMC_Nf2p1_eo
a : Test All
p : Setup test check precision (current precision: 12)
u : Go back
q : Quit
choice> 1
```

SOLVER & FERMION OPERATOR

- One can change a fermion operator and a linear solver independently.
- Same mechanism is used in smearing.

MULTI-LEVEL HMC INTEGRATOR

- Multi-level leapfrog:
  - $U_0(t) = \left[P_k\left(\frac{\Delta\tau_0}{2}\right) Q(\Delta\tau_0) P_0\left(\frac{\Delta\tau_0}{2}\right)\right]^{N_0}$,
  - $U_k(t) = \left[P_k\left(\frac{\Delta\tau_k}{2}\right) U_{k-1}(\Delta\tau_k) P_k\left(\frac{\Delta\tau_k}{2}\right)\right]^{N_k}, \Delta\tau_k = \frac{t}{N_k}$
- Same mechanism is used in fermion operator.



WIKI DOCUMENTATIONS

本文 | 議論 | ソースの表示 | 以前のリビジョン

Bridge++ wik

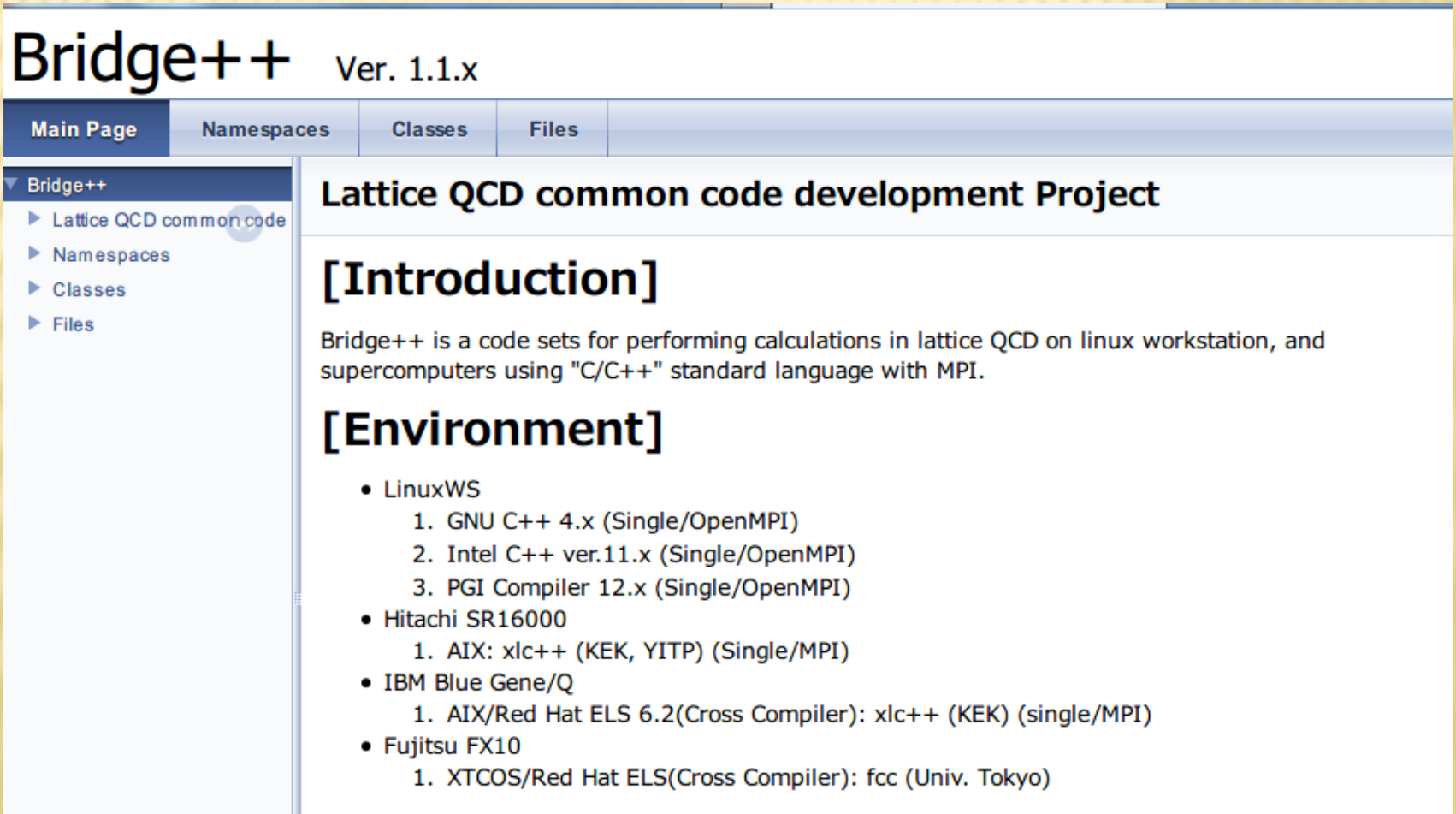
Bridge++ wiki English index page

Welcome to Bridge++ Wiki: documents for Lattice gauge theory simulation code Bridge++.

- Official page: http://bridge.kek.jp/Lattice-code/index_e.html
- Policy of development
- Current status
- Version information
- First step guide
- Code implementation guide
- Code implementation (details and extension)
- Notice for each environment
- Confirmation information
- Bug report/feedback
- Acknowledgment

DOXYGEN DOCUMENT

- HTML link base document
- Generated from code comments



CODE TUNING (IN PROGRESS)

We have started machine specific tuning for example on BG/Q.

- Wilson mult (16^3x32, 32nodes):
  - OpenMP: 25.1 GFlops (12-13%)
  - Pthread: 25.5 GFlops (12-13%)
  - BG Wilson Lib: 37.6 GFlops (17-18%)
- Solver:
  - OpenMP: 23.7 GFlops (11-12%)
  - BG Wilson Lib: 26.1 GFlops (13-14%)

SUMMARY

Lattice code “Bridge++”

- C++, Object oriented
- Readability, Extensibility, Portability, High-performance

Still being actively developed

- Refactoring and implementing new functions
- Optimizing to BG/Q, SR-16K, K-computer, GPU, Xeon phi

Please use “Bridge++”
and give us comments for feedback.

BACKUP SLIDE

CODE VIEW: SOLVER

```
192  void Solver_CG::solve_step(double& rr)
193  {
195    m_fopr->mult(s, p);
196
197    double pap  = p * s;
198    double rr_p = rr;
199    double cr   = rr_p / pap;
200
201    v  = p;
202    v *= cr;
203    x += v;
205
206    s *= cr;
207    r -= s;
208
209    rr = r * r;
210    p *= rr / rr_p;
211    p += r;
212  }
```

CODE VIEW: INTEGRATOR

```
90  void Integrator_Leapfrog::evolve(Field_G& iP,
    Field_G& U)
91  {
96    // set up phase
105
106   // Initial half step of update of iP
107   if (m_Nstep > 0) {
108     int istep = 0;
109     vout.general(m_vl, "istep = %d¥n", istep);
110     force = 0.0;
111     for (int i = 0; i < m_action.size(); ++i) {
112       force1 = m_action[i]->force();
113       force += esteph * force1;
114     }
115     iP += (Field_G)force;
116   }
118   // Molecular dynamics step
119   for (int istep = 1; istep < m_Nstep + 1;
    istep++) {
121
122     m_integ_next->evolve(iP, U);
123
124     estep2 = estep;
125     if (istep == m_Nstep) estep2 = esteph;
126
127     force = 0.0;
128     for (int i = 0; i < m_action.size(); ++i) {
129       force1 = m_action[i]->force();
130       force += estep2 * force1;
131     }
132     iP += (Field_G)force;
133   } // here istep loop ends
134
136 }
```