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

UEDA Satoru (KEK)  
Bridge++ Project

Lattice 2013 @ Mainz, Germany August 2, 2013
We are developing a lattice QCD code set Bridge++

- Project site:
  - [http://bridge.kek.jp/Lattice-code/](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:
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 has been held every 1-2weeks
  - Advices given by experts in computer science and applied mathematics
- 2012 July: ver.1.0 public released
- 2013 23rd July: ver.1.1 released
OUTLINE

- Introduction
- Implementation sample:
  + Solver & fermion operator
  + HMC integrator
- Documentation
- Code tuning (In progress)
- Summary
Recent Lattice simulations require:

- Various physical models (beyond SM etc.)
- Variety of architectures (massively parallel multi-level processor, 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, OpenMP/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 etc.
  - ILDG configuration format
  - YAML parameter file

- Now being confirmed:
  - Quark number Susceptibility

- Being developed:
  - OpenMP/pthread, OpenCL, CUDA
About 40 test modules are provided.

- Implementation samples, how to use the classes
- Verification tool
- Interactive test manager
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.
First step guide
  + How to compile and run

Implementation note
  + Memos, write when we implement classes

Verification reports
  + Data of compared with previous papers
[Introduction]

Bridge++ is a code set for performing calculations in lattice QCD on Linux workstation, and supercomputers using "C/C++" standard language with MPI.

[Environment]

- LinuxWS
  1. GNU C++ 4.x (Single/OpenMPI)
  2. Intel C++ ver.11.x (Single/OpenMPI)
  3. PGI Compiler 12.x (Single/OpenMPI)
- Hitachi SR16000
  1. AIX: xlc++ (KEK, YITP) (Single/MPI)
- IBM Blue Gene/Q
  1. AIX/Red Hat ELS 6.2 (Cross Compiler): xlc++ (KEK) (single/MPI)
  2. Fujitsu FX10
  1. XTOS/Red Hat ELS (Cross Compiler): fcc (Univ. Tokyo)
We have started machine specific tuning for example on BG/Q.

- Wilson mult:
  - OpenMP: 25.5 Gflops (12-13%)
  - BG Wilson Lib: 37.6 Gflops (17-18%)

- Solver:
  - OpenMP: 25.7 Gflops (12-13%)
  - 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.
```cpp
void Solver.CG::solve_step(double& rr)
{
    m_fopr->mult(s, p);

double pap = p * s;
double rr_p = rr;
double cr = rr_p / pap;

    v = p;
    v *= cr;
    x += v;

    s *= cr;
    r -= s;

rr = r * r;
p *= rr / rr_p;
p += r;
}
```
```cpp
void Integrator_Leapfrog::evolve(Field_G& iP, Field_G& U) {
    // set up phase
    // Initial half step of update of iP
    if (m_Nstep > 0) {
        int istep = 0;
        vout.general(m_vl, "istep = %d\n", istep);
        force = 0.0;
        for (int i = 0; i < m_action.size(); ++i) {
            force1 = m_action[i]->force();
            force += esteph * force1;
        }
        iP += (Field_G)force;
    }
    // Molecular dynamics step
    for (int istep = 1; istep < m_Nstep + 1; istep++) {
        m_integ_next->evolve(iP, U);
        estep2 = estep;
        if (istep == m_Nstep) estep2 = esteph;
        force = 0.0;
        for (int i = 0; i < m_action.size(); ++i) {
            force1 = m_action[i]->force();
            force += estep2 * force1;
        }
        iP += (Field_G)force;
    } // here istep loop ends
} // Integrator_Leapfrog::evolve
```