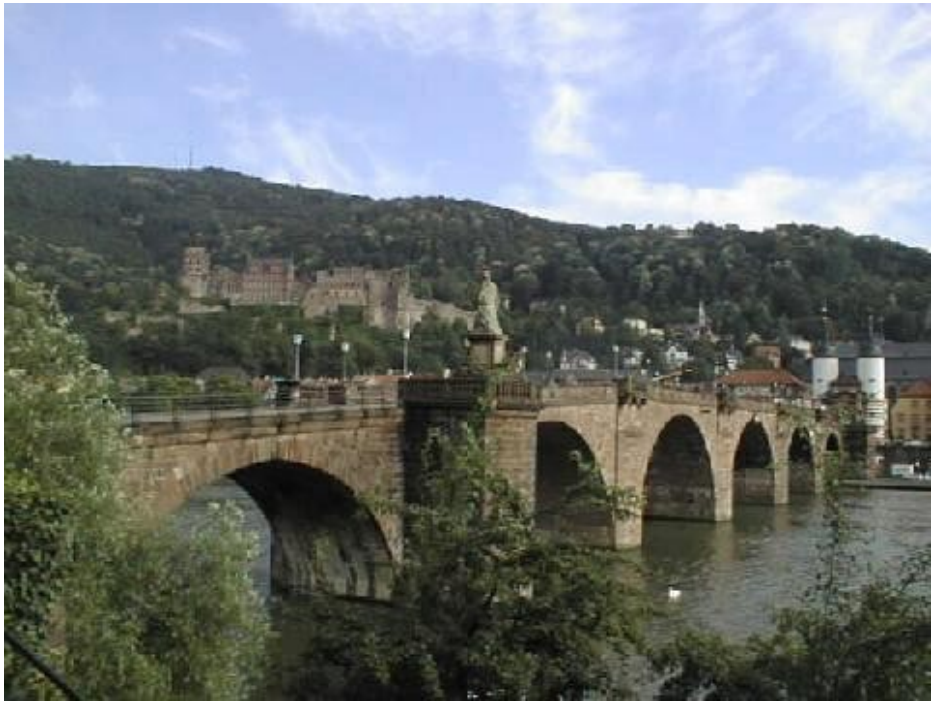
Bridge++の実装
Gauge fixing



22 Feb 2012

Hideo Matsufuru (KEK)

進捗

新機能

- Landau gauge fixing: gaugeFixing_Landau class
- Coulomb gauge fixing: gaugeFixing_Coulomb class
- Smeared source

新しい環境でのチェック

- NEC SX-8/9 at RCNP

メモ

- 例外についての所見

Gauge fixing

- Hadron operator smearing, nonperturbative renormalization などで必要

実装

- Los Alamos 法 (? 何と呼ぶのが正確？)による実装
- 基本原理はLandauとCoulombで同じ
- Overrelaxation
- Gribov copy: random gauge transf. で脱出
- 現在は SU(3)のみに適用可
- 継承構造は今のところ無し (fix するゲージが流動的要素となるケ場合を思いつかないので)

おまけ

- Exp smeared source
  - Local source とともに、継承無しのバージョンを新規作成
  - 検証はまだ

Landau gauge fixing の実装

- 基本原理はLandauとCoulombで同じ
- 青木さんの教科書参照（ただしちょっと違うアルゴリズムかも）

原理：

$$F[U] = \sum_{x,\mu} \mathrm{ReTr} U_\mu(x)$$

という量をゲージ変換によって最大化する。

$$U_\mu(x) \to G(x) U_\mu(x) G^\dagger(x+\hat{\mu})$$

次の量 w(x) を定義すると、$F[U] = \frac{1}{2}\mathrm{ReTr}\sum_x w(x)$

$$w(x) = \sum_\mu [U_\mu(x) + U_\mu^\dagger(x-\hat{\mu})]$$

even-odd にサイトを分け、even-site のみでの変換を考えると、

$$w(x) \to G(x) w(x)$$

Landau gauge fixing の実装

従って次の条件を満たすゲージ変換 G(x) を作ればよい

$$\mathrm{ReTr}G(x)w(x) \geq w(x)$$

このような変換行列は、SU(2)部分群ごとに

$$\mathrm{ReTr}G(x)w(x)$$

を最大化することによって作ることができる（Cabibbo-Marinari）

Coulomb gauge の場合

- 各 time slice で Landau と同じアルゴリズムを適用

これらを、次のクラスで実装した（継承構造は今のところ無し）

- GaugeFixing_Landau
- GaugeFixing_Coulomb

Check

Landau, Coulomb それぞれについて以下のチェック

- Gauge invariance
  - Plaquette, meson correlator が良い精度で一致を確認
- Compiler 依存性
  - g++, intel, SX, XL-C (並列)で一致を確認
- 並列化
  - 2x2x2x4-MPI分割、4^3x8, 8^3x16 lattice で確認
  - Even-odd なので、ノード内で奇数になるサイズも試しておきたい
    → smeared meson で oven-odd clover と同時に出来れば better
- Fortran code とのチェック
  - Fortran code: JLQCD overlap project で使ったもの
  - C++ code で作った fixed config を読み込んで、収束していることを確認
- 最終的には smeared correlator を文献値と合わせるのが best
- Smeared source: Source_4spinor_exp で実装

Gauge fixing のやり残し

- 一般の SU(N) での Tr[UV^¥dag] の maxmization
  - SUNmat class に組み込むのが良いか?
  - できれば det(), reunit() も同時に
- Spatial site をまとめて扱う仕組み
  - Coulomb では timeslice 内での Field の演算が多発
    → x,y,z,t でループを回すのが煩雑＆余分な演算
- Performance は多分イマイチ
  - 一度 SU(N) のインスタンスにする部分が散在
  - それほどスピードが必要な訳ではないが、何もしなくてもそこそこのスピードが出て欲しい

On SX

SX@RCNPの上で動かしてみた

- 1CPU job, SX8R, 32GFlops peak
- SX compiler, 最適化＋ベクトル化オプション
- 例外が default では不可、要オプション → throw は全部外した
- Binary I/O: C の system callを使う部分がコンパイルエラーになったので、実装を変更 (Stroustrap's book に従った)

- Code: Bridge-OF 0.9.293 (repository版とほぼ同じはず)
  - Gauge fixing (L/C) + clover solver + meson correlator
  - Shift, Field_G, Field_F, Wilson, clover は imp-version
  - 4^3 x 8 lattice で他の環境の結果と一致を確認
  - 他の機能については未検証
- Performance
  - 4^3 x 8, gfix+solver+meson: 0.14 Gflops (v.op 92%)
  - 4^3 x 8, solverのみ: 0.33GFlops (v.op 67%)
  - 8^3 x 16, solver のみ: 0.31GFlops (v.op 57%)
  - かなり悪い (ToT)

例外についての所見

例外処理は必要か？

- 野秋さん「例外安全のスゝメ」 30 Jul 2010
  - 例外安全：エラーが生じた場合でもプログラムの状態を正常に保ち、リソースもれを防ぐ
  - 動的なメモリ確保を行うもの（STLコンテナの全てを含む）
    →失敗したとき、bad-alloc 例外を投げる
- More Effective C++: Item 15
  - try block の使用によって、exceptionが投げられないときでも 5-10% のコストアップ
  - 不要な try block は使わないことを推奨
- Effective C++
- SX compiler
  - default では例外を disable, option指定が必要
  - 例外の監視、例外の送出のために命令列がオブジェクトファイル内に生成され、実行される。命令列が著しく増加する場合がある

例外についての所見

- Stroustrap（プログラミング言語 C++ Sec.14)：
  - 基本的な考え方は、自分で対処できない問題を見つけた関数は、(直接、間接の)呼び出し側コードが問題に対処することを期待して、例外を投げる: throw というものである。」
- Lattice code の場合、どういう対処があり得るか？
  - ゼロ割、underflow/overflow, file open の失敗、メモリなどのリソース確保失敗、not converged, etc. が「例外的事象」
  - それでも処理を続ける必要はない、むしろそこで実行を中止してくれる方が状況を理解しやすい
  - どこで中止したかは重要な情報
- 私見
  - パフォーマンスコストの可能性を考慮してまで例外が必要な状況は、ほとんど無い

    → 提案： Farewell to exceptions: dying message を残してその場でabort()