Nambu-Bethe-Salpeter 波動関数を計算するコードの Bridge++への実装

(第 2 版)

根村英克， 筑波大学

*October*6, 2012

1 はじめに

原子核物理の基本的な問題の一つは、原子核を記述するために基本的な自由度として扱われる核子（バリオン）の間にはたらく相互作用を解明することである。とりわけ、中性子星の内部のような高密度状態の構造や、エキゾチックハドロンの候補として考えられている H-ダイバリオン状態への関心が高まっており、バリオンとして核子だけでなくハイペロンを含んだ場合の相互作用をより詳しく調べることが重要となっている。

核力を記述する現象論的模型では、そこに含まれているパラメータを、豊富な核子散乱および重陽子のデータを利用して決めることがなされているいっぽうで、そのような核力模型をハイペロンを含んだ場合へ拡張しようとすると、ハイペロン散乱のデータが核子散乱に比べて限られているため、核力に匹敵する精度でハイペロン相互作用を記述することは難しい。

精密に決められた核力を用いて、軽い原子核の結合エネルギーがどの程度再現できるかという議論が行われているが、これまでのところ現象論的に決められたハイペロン相互作用を用いて軽いハイパー核の結合エネルギーを再現しようとすると、必ずしもうまく行かないことが示されている。これは、ハイペロン散乱やハイパー核の実験が難しく、ハイペロン相互作用の性質を定量的に決定するには、実験データがまだじゅうぶんに揃っているとはいえないためと考えられる。

最近になって、格子 QCD 計算から、複数のハドロン間にはたらく相互作用を調べる研究がさかんに行われるようになってきた。とりわけ、格子 QCD から核力ポテンシャルを求める研究が行われ始め、ハイペロンへポテンシャルへの拡張も進められている。これらの計算は、格子 QCD 計算としては標準的な部分（ゲージ配位生成、クォーク伝搬関数の計算）と、核力・ハイペロンポテンシャルを求めるために新たに付け加えられた部分（Nambu-Bethe-Salpeter (NBS) 波動関数の計算）を組み合わせて行われている。

これまでの研究では、主に、ゲージ配位生成は独自に行わず、ILDG/JLDG によって公開されているゲージ配位を利用し、クォーク伝搬関数の計算部分は CPS++[1] をもとにしたコードセットを利用し、CG 法および BiCGStab 法について HAL QCD collaboration で必要な拡張を加えた上で、NBS 波動関数の計算部分を付け加えたものを、大型計算機の上で動かしている。

最近、格子 QCD の共通コードセットとして、Bridge++ が公開されている [2]。筆者はこの Bridge++ の主要な開発メンバではないが、Bridge++ 開発は、JICFUS の活動のひとつでもあり、今後これを利用して研究をすすめていければと考えて開発ミーティングなどに参加させて頂いている。

本来、ハイペロンポテンシャルを求めるための NBS 波動関数を計算する部分は、CPS++ とは独立しているので、CPS++ に依存せず、別のコードセット（例えば Bridge++）と組み合わせて計算できるようにコードの構造を整理しておくことは、将来的な発展を考える上でメリットがあると思われる。

この報告書では、CPS++ と組み合わせて NBS 波動関数を計算するように作られている現在の C++ コードを、Bridge++ と組み合わせても動くように整備したので、その概要についてまとめる。

2　Nambu-Bethe-Salpeter 波動関数の計算の概要

詳細は別の文献 [3] に譲ることにして、ハイペロンポテンシャルを研究するために、大型計算機を用いて計算しているのは、次のような格子 QCD の 4 点相関関数である。

$$F^{(J,M)}_{\alpha\beta}(\vec{r}, t-t_0) = \sum_{\vec{X}} \left\langle 0 \left| B_{1,\alpha}(\vec{X}+\vec{r}, t) B_{2,\beta}(\vec{X}, t) \overline{\mathcal{J}^{(J,M)}_{B_1B_2}(t_0)} \right| 0 \right\rangle . \tag{1}$$

$\vec{X}$ についての和は、全系の運動量をゼロとするためにとられている。演算子 $B_{1,\alpha}(x)$ および $B_{2,\beta}(y)$ は計算しようとしている系のバリオンと同じ量子数を持つように決められる。ここでは、陽子およびラムダ粒子について、以下のものを使っている。

$$p_\alpha(x) = \varepsilon_{abc} \left(u_a(x) C\gamma_5 d_b(x)\right) u_{c\alpha}(x), \tag{2}$$

$$\begin{aligned}\Lambda_\beta(y) = \varepsilon_{abc} \{ &(d_a(y) C\gamma_5 s_b(y)) u_{c\beta}(y) + (s_a(y) C\gamma_5 u_b(y)) d_{c\beta}(y) \\ &-2(u_a(y) C\gamma_5 d_b(y)) s_{c\beta}(y)\} .\end{aligned} \tag{3}$$

演算子 $\overline{\mathcal{J}^{(J,M)}_{B_1B_2}(t_0)} = \sum_{\alpha'\beta'} P^{(J,M)}_{\alpha'\beta'} \overline{B_{1,\alpha'}(t_0) B_{2,\beta'}(t_0)}$ は、計算しようとしている系 B_1B_2 の、角運動量 (J, M) 状態を生成する演算子である。

3　CPS++ から Bridge++ への移植作業の概要

当然ながら計算の主要部分は変更する必要はなく、CPS++ に依存する書き方をしている部分を、より一般的な書き方に変更する、もしくは CPS++ を使っているか、Bridge++ を使っているかを切り替えられるように変更する、という作業を行うことになる。もう少し具体的な点を以下にまとめる。

[1] 4 点相関関数の計算は、ハドロンスペクトルを計算するための 2 点相関関数の計算の拡張のようなものなので、NBS 波動関数を計算する部分のソースコード一式は、`src/Measurements/Fermion/NBSWF/` にまとめて配置する形にした。

[2] CPS++ では拡張子を `.C` としていたが、Bridge++ ではデフォルトの設定では `.C` ファイルは無視されるので、拡張子を `.cpp` とする。作業量を減らすために、今回は、これを逆に利用して、手で直接ソースファイルを編集する必要がある場合のみ、`.C` ファイルを編集し、`.C` から `.cpp` へのコピーおよび機械的な作業で行える変更は、`make` を掛ける直前で、スクリプトなどで処理する方針とした。`.C` ファイルを編集する際には、CPS++ へ再び戻りたい場合のことを考えて、この `.C` ファイルを含むソースコードセットを再び CPS++ と組み合わせても動くように配慮 (`#ifdef` ⋯ `#else` ⋯ `#endif` を利用) して変更を加えた。

[3] メモリの動的確保および解放に `malloc`, `free` を使っていた部分は、C++ で提供されている `new`, `delete` に置き換える。

[4] 並列計算において格子 QCD 計算の基本パラメータを受け取っている部分は、CPS++ の実装に加えて Bridge++ 用の実装を追加する。今回必要だったものを、表 1 にまとめる。

[5] 計算の途中結果などを出力している部分は、CPS++ の実装に加えて、Bridge++ 用の実装を追加する。Bridge++ での典型的な書式は、

```
vout.important(m_vl, "This is a verbose output: %d\n", i);
```

Table 1: NBS 波動関数の計算を実装するために必要な CPS++ と Bridge++ での格子 QCD の基本パラメータの対応表。

CPS++	Bridge++(preferred)	Bridge++(alias)
`GJP.Xnodes()`	`Communicator_impl::Layout::npe(0)`	`CommonParameters::NPEx()`
`GJP.Ynodes()`	`Communicator_impl::Layout::npe(1)`	`CommonParameters::NPEy()`
`GJP.Znodes()`	`Communicator_impl::Layout::npe(2)`	`CommonParameters::NPEz()`
`GJP.Tnodes()`	`Communicator_impl::Layout::npe(3)`	`CommonParameters::NPEt()`
`GJP.XnodeSites()`	`CommonParameters::Nx()`	
`GJP.YnodeSites()`	`CommonParameters::Ny()`	
`GJP.ZnodeSites()`	`CommonParameters::Nz()`	
`GJP.TnodeSites()`	`CommonParameters::Nt()`	
`GJP.XnodeCoor()`	`Communicator_impl::Layout::ipe(0)`	
`GJP.YnodeCoor()`	`Communicator_impl::Layout::ipe(1)`	
`GJP.ZnodeCoor()`	`Communicator_impl::Layout::ipe(2)`	
`GJP.TnodeCoor()`	`Communicator_impl::Layout::ipe(3)`	

[6] CPS++ に含まれているヘッダファイルを `#include` している部分は、適切に置き換える。

[7] 4 点相関関数の計算を高速に行うために、Fast Fourie Transform (FFT) を用いているので、コンパイル時に必要なファイルへのパスが適切に通るようにしておく。

[8] 上記変更などが適切に反映されるように、Makefile を修正する。

4 まとめ

これまで、CPS++ と組み合わせて動かしていた NBS 波動関数の計算コードを、Bridge++ と組み合わせても動くように、実装を行った。

Bridge++ は、ノート PC のような個人の計算機から共同利用されている大型計算機まで、様々な環境の上で効率よく動くようにするための開発が現在も継続されている。今後は、パフォーマンスチューニングを行いつつ、本格的な計算のための準備を進めて、Bridge++ と組み合わせた NBS 波動関数の計算結果を用いて新しい成果が出せるところまで進めていきたい。

References

[1] Columbia Physics System (CPS) [`http://qcdoc.phys.columbia.edu/cps.html`].

[2] Lattice QCD 共通コードプロジェクト [`http://suchix.kek.jp/bridge/Lattice-code/`].

[3] H. Nemura [HAL QCD and PACS-CS Collaborations], PoS LAT **2009**, 152 (2009) [arXiv:1005.5352 [hep-lat]].