Vol.2012-HPC-136 No.2
2012/10/3

C++コードに対する OpenCL を利用したアクセラレータフレームワークの開発

元木 伸治[1,a)]　松古 栄夫[2,b)]

概要：本稿では, 近年 PC ワークステーションからスーパーコンピュータまで, 目覚ましい発展を遂げているメニーコアシステム上における一般的な計算フレームワークの開発の試みについて考察した. これは, 我々の開発している QCD(Quantum Chromodynamics, 量子色力学) を含む格子ゲージ理論のシミュレーションのためのコードセット Bridge++において, ヘテロジーニアス環境, メニーコア環境を組み込むための, 設計方針の研究である. アプリケーションの構築環境（いわゆる SDK) に可能な限り依存しない実装を実現するため、OpenCL プラットフォーム規格を用いて, 演算アクセラレータ利用のための一般的フレームワークについて検討を行った.

1. はじめに

単一 CPU の性能は 2004 年頃から頭打ちとなっているが, 2006 年 8 月の CPU カンファレンス「HotChips 18」において David A. Patterson は, その理由を下記に挙げる 3 つの壁として説明した [2].

- 命令レベルの並列性である「ILP(Instruction-Level Parallelism)」の限界
- 消費電力の壁「Power Wall」
- メモリアクセスの壁「Memory Wall」

である. その後, 汎用プロセッサと特定演算に特化したプロセッサを組み合わせたヘテロジーニアス環境が盛んに研究されてきた [3], [4], [5], [6], [7], [8], [9], [10], [11]. いわゆる汎用 CPU+アクセラレータという考え方である. アクセラレータは, 回路が単純なため, 集積率を高くできると共に, 消費電力を押さえる事ができるという利点もある.

このように, アクセラレータとの連携によって演算の高速化が期待されるが, 利点ばかりではない. 「Memory wall」の問題のように演算が高速化されたが故にメモリバンド幅が律速となる傾向がさらに顕著になったのも事実である. スーパーコンピュタはこの辺の事情を専用アーキテクチャである程度克服しているが, 個人の研究室で保有可能なクラスター環境や, GPU を搭載した PC ワークステーションではなかなかそうはいかない. そのような環境下で, 最適化したコードを記述しようとすると, とたんにコードの可搬性が失われてしまう. つまり, 計算機が変わるたびにコードの書き換えが必要となってしまう. 更に, 十分な性能を得るためにはアルゴリズムの変更が必要な場合もある. これらは簡単な作業ではないため, 本来の目的である新しいアイデアや新しい物理量を試すのが難しい, という状況に陥ってしまうことになりかねない.

以上のような状況を鑑みて, 我々は 2009 年より, 格子ゲージ理論のシミュレーションのための共通コード開発プロジェクトを進めている. C++を用いてオブジェクト指向に沿ったデザインを採用し, 利用者にとっての可読性を重視すると同時に, 実用に耐えられるようなパフォーマンスを持つコードを目指している. マシンに依存する実装は可能な限り減らすかもしくは隠蔽し, 様々なアルゴリズムを試してゆくにあたって便利な構造とする必要がある. 今回の試みは, 本コードセットにおいてアクセラレータを利用するために必要なデザインを検討するためのものである. 可能な限り特定のハードウェアに依存しない実装を実現するために, OpenCL プラットフォーム規格を採用し, アクセラレータを計算リソースとして一般的に扱う. 他分野での計算にも適用可能なフレームワークとしての一般化も目指している.

本稿は以下のような構成である. まず次章で, 格子ゲージ理論のシミュレーションについて紹介する. 特に, シミュレーションにおいて最も時間のかかり, アクセラレータによる高速化の最大のターゲットである, 大規模疎行列の線形方程式の問題について述べる. 3 章では, 我々の開発し

[1] 高エネルギー加速器研究機構素粒子原子核研究所理論センター
1–1 Oho, Tsukuba, Ibaraki 305-0801 Japan
[2] 高エネルギー加速器研究機構計算科学センター
1–1 Oho, Tsukuba, Ibaraki 305-0801 Japan
[a)] smotoki@post.kek.jp
[b)] hideo.matsufuru@kek.jp

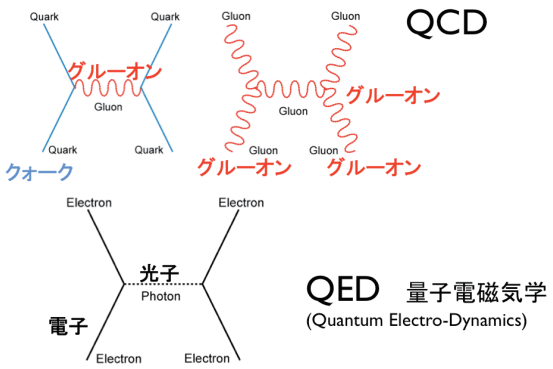

図 1 QED と QCD の概念図

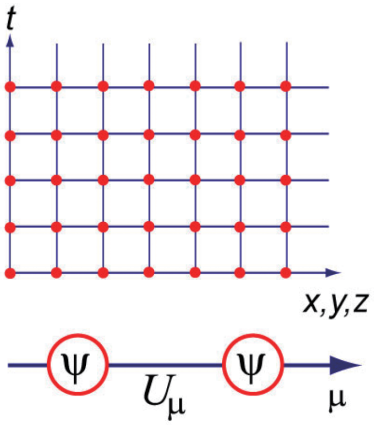

図 2 格子上でグルーオン場 U_μ はリンクの上に定義され, クォーク場 Ψ は格子点上に定義される

ている格子 QCD シミュレーションのためのコードセット Bridge++について紹介する．4 章では OpenCL 規格についてまとめ, 5 章でこれを用いてアクセラレータを利用するためのデザインを検討する．

2. 格子 QCD シミュレーション

2.1 格子 QCD とは

陽子や中性子を構成する素粒子であるクォークの間の相互作用は, グルーオンと呼ばれる粒子によって媒介され, 量子色力学 (Quantum Chromodynamics, QCD) によって記述される．QCD は, 電子が光子を介して相互作用する量子電気力学 (Quantum Electrodynamics, QED) に似た, 自由度が場の量として表される理論体系である．このように力を媒介する粒子の場を, ゲージ場と呼ぶ．QED と QCD の本質的な違いは,QED が可換な U(1) 群 (1 次元ユニタリー群) に基づくのに対し, QCD は非可換群の SU(3) 群 (3 次元特殊ユニタリー群) に従って構成されることであり, その非可換性からグルーオン同士の相互作用が存在することである (図 1)．QED では電子と格子の間の結合が小さく, 結合定数による展開である「摂動論」が大きな成功を収めたが,QCD では結合が強いために摂動論は高エネルギー領域以外では使えない．得に, クォークが単体で観測されない「閉じ込め」という現象など, QCD の持つ特異な性質を解明するためには, 何らかの非摂動的な手法が必要である．

QCD の定量的研究のための非常に強力なツールは,K.G. ウィルソンによって提案された, 離散化されたユークリッド空間, つまり格子上での場の理論 (格子 QCD) である [22]. QCD ではゲージ場 (グルーオン) は SU(3) 群の元であり，3×3 の複素行列によって表され，4 次元ユークリッド格子の，格子点と格子点を結ぶ “リンク” の上に定義される (リンク変数とも呼ばれる) (図 2)．従って, サイトと方向の自由度を持つため、$U_\mu(n)$ のように表される．ここで $n=(n_x,n_y,n_z,n_t)$ は格子点を表し, シミュレーションにおいては, 格子の大きさは有限であるので, $n_x=1,2,\cdots,N_x$, などとなる．クォーク場は格子点 (サイト) 上に定義され, 3 成分の色の自由度と 4 成分のスピノルの自由度を持つ, $3\times4=12$ 成分の複素ベクトルとして表される．

格子 QCD は経路積分法によって量子化される．経路積分では，クォーク場は反交換関係を満たす数として定式化されるが，このような数を計算機上で直接用いるのは有効ではない．実際に様々な物理量を計算するために必要なのはクォークの伝播関数である．伝播関数はある格子点から別の格子点への伝播を表し, クォークの演算子 (巨大疎行列) の逆行列として定義される．例えばハドロンの質量は, クォークの伝播関数から構成されるハドロンの相関関数の振る舞いから引き出すことができる．数値シミュレーションでは，ゲージ場の配位 $\{U\}$ を Monte Carlo 法によって確率的に生成し，それらの配位の上で様々な相関関数を計算することによって，物理量を求めることが可能となる．その際, クォークの効果は, 伝播関数を通して寄与するため,Monte Carlo 法の各ステップでクォーク演算子に対する線形方程式を解く必要が生じる．これが格子 QCD シミュレーションにおいて最も時間のかかる部分である．

2.2 格子 QCD シミュレーションにおける線形問題

クォーク演算子としては, 様々な形のものが提案されている。例として, 最も基本的な形である, ウィルソン (Wilson) によって提唱されたものを示す．

$$D(m,n)=\delta_{m,n}-\kappa\sum_{\mu=1}^{4}[(1-\gamma_\mu)U_\mu(m)\delta_{m+\hat{\mu},n} +(1+\gamma_\mu)U_\mu(m-\hat{\mu})\delta_{m-\hat{\mu},n}] \quad (1)$$

ここで,m,n は格子点, $\hat{\mu}$ は μ 方向の単位ベクトル, κ はクォークの質量に関係するパラメータ,γ_μ はスピノル空間に作用する 4×4 の行列である。クロネッカーのデルタの構造からわかるように, この演算子は隣合うサイト間の結合しかないため, 行列としては次数 $3\times N_x\times N_y\times N_z\times N_t$ の巨大な複素疎行列となる (図 3)．Wilson 演算子以外の場

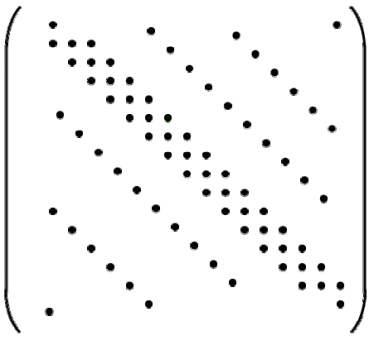

図 3 演算に現れる疎行列のイメージ

合, 結合が最隣接サイトだけでない場合もあるが, 一般的には疎行列の線形問題に帰着できる.

伝播関数はクォーク演算子の逆であり $[S_q(m,n) = D^{-1}(m,n)]$, 点 n を固定して

$$D(m,l)S_q(l,n) = \delta_{m,n} \tag{2}$$

という線形方程式を,CG 法のような反復法によって求める. 反復法のカーネルである行列ベクトル積の部分は，クォーク演算子とクォーク場ベクトルの積となる．格子 QCD シミュレーションでは，クォークの伝播関数の計算が計算時間のほとんどを占めるため，この演算が高速化の鍵となる.

3. 共通コードプロジェクト

3.1 開発の経緯とその背景

格子 QCD は計算機の高速化と共に発展し, 以前は難しかった現実的クォーク質量でのシミュレーション (大きな格子が必要かつ反復法のコストが大きいため) が最近では可能になってきた. 一方でアクセラレータの発達などにより, 以前はスーパーコンピュータでしかできなかった計算が手軽にできるようになった. このような計算環境の多様化は, 研究者の選択肢を増やし大規模計算を身近にしたが, 一方でハードウェアの多様性とチューニングの必要性から, コードは複雑化した. このため, 研究グループを変える毎にコードを勉強しなければならない, 計算機に合わせてコードを書き直さなければならない, 初心者がコードを勉強するのが難しい, などの問題が顕著になってきた. 特に新しいアイデアや新しい物理量を手軽に試せないことは大きな問題である.

このため, 格子 QCD の共通コードが公開されてきた: MILC コード (C, 米国)[12], CPS++ (C++, 米国)[13], Chroma(C++, 欧州)[14], Lattice QCD toolkit(Fortran, 日本)[15], などである。しかしながら、日本はアメリカ, ヨーロッパと並ぶ格子 QCD の拠点の１つであるにも関わらず, Lattice QCD toolkit 以外の本格的な共通コードが無いことは, 将来の研究体制に問題であるとの認識があった. コード改変のフィードバックや計算機固有の最適化の時間差をなくし, コードの管理とユーザに対するサポート体制を国内で整備することには大きな意義がある. これらの理由から, 新領域研究「素核宇宙連携による計算基礎科学に基づいた重層的物質構造の解明」(領域代表：青木慎也) において, 格子ゲージ理論のシミュレーションのための共通コードプロジェクトが 2009 年より開始された.

3.2 共通コード (bridge++) の特徴

本共通コードでは, 前述の背景を踏まえ, 以下の点を目標に挙げ, 開発を進めている.

可読性: 初心者が読んでも理解できること
拡張性: 新しい機能を組み込みやすいこと
高性能: 実戦に十分なパフォーマンスを持つこと
開発上の方針: C++による実装

- オブジェクト指向に基づいたデザイン
- デザイン・パターンなど, ソフトウェア工学上の知見を活用
- 十分なパフォーマンスの実現

可搬性: 様々な環境に対応
ノート PC での開発から, スパコンでの本格ランにも対応 MPI 並列化,(MPI 環境のない) シングルノードにも対応

ユーザへのアンケート, 既存コードシステムの調査などを経て実装を開始し, 2012 年 7 月 23 日に最初の公開版をリリースした [28]. また、このようプロセスおよび設計の手続きは, アメリカのライス大学等で発展した計算科学分野の一分野である Problem Solving Environment:問題解決環境 (PSE) の試みとしても位置づけることもできる [21]. コードセットは現在はまだ基本機能の公開にとどまっているが, クォーク演算子等の種類を増やし, アルゴリズムや物理量を充実させると共に, 各計算機でのパフォーマンスを上げるための開発が続けられている. 中でも,GPGPU などのアクセラレータを利用するための仕組みを組み込むことは喫緊の課題の一つである.

4. OpenCL フレームワーク

OpenCL プラットフォーム規格は, OpenGL でも有名な非営利団体 Khronos Group[1] が仕様の標準化を行い, AMD,

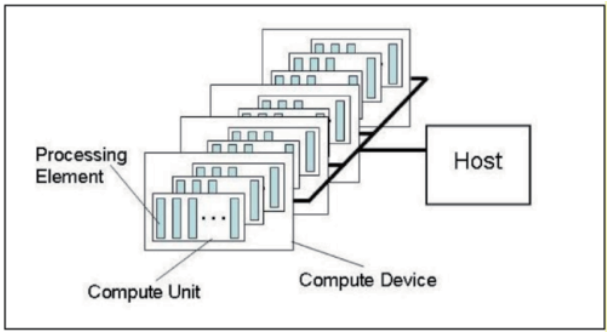

図 4 OpenCL プラットフォームモデル概念図 [1]

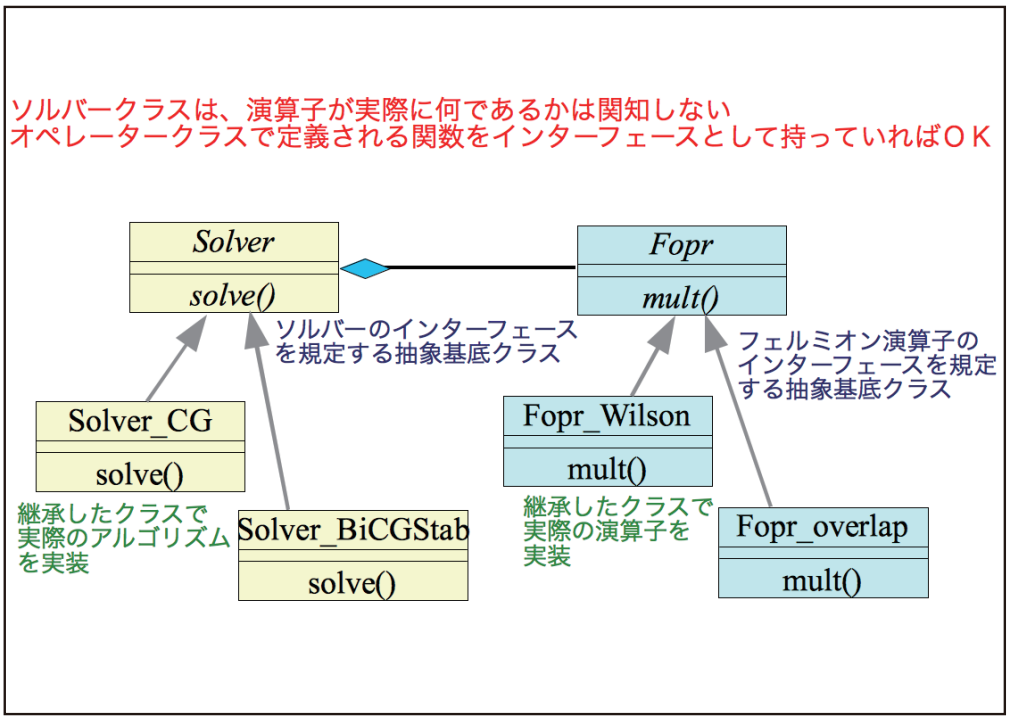

図 5 コードセットにおけるクォークプロパゲータ演算の実装設計

Apple, IBM, NVIDIA, Texas Instruments, Sony, Toshiba などが趣旨に賛同し参加している. 以下では,OpenCL の基本的な考え方について簡単に述べる.

OpenCL のアプリケーションは, ホストプログラムと複数あるいは単一のコンピューティングデバイスから構成される. 一般的にこれらを総称してプラットフォームと呼んでいる. ホストデバイスはメモリアクセスをコントロールし,GPU や CPU あるいは, その他の利用可能なデバイス上での演算の指示, 実行を行う. また, 対象とするプラットフォームが, ホモジーニアスな環境 (いわゆるマルチコア CPU の場合) であったとしても, 一つのコアがホストを担当し, その他のコアが実際の演算デバイスとして扱われ,OpenCL を用いた実行が可能である. OpenCL のプラットフォームモデルを図 4 に示す.

OpenCL の, 最大の特徴は並列計算機を構成するプロセッサの種類やベンダに依存することなく, 統一された並列ソフトウェア開発環境を実現する為の仕様が標準化されている点である. ソフトウェア開発者は OpenCL によって標準化された言語と API を用いる事で, チップや OS に依存しない並列処理ソフトウェアを開発できるようになる. これが我々が重視する要素の一つである可搬性に欠かせない点である. このように, 多彩な環境に対応し, 覚えなければならないプログラム上の文法が一つで済む利点は大きい. また, コード記述を一般化する事のトレードオフとしてパフォーマンスが懸念されるが, OpenCL では低レベル API 部分での環境に応じた高速化も可能であり, 高速化が必要な部分に関しては, それぞれの環境用のカーネルを用意する事も可能である. また, 既知の通り,OpenACC[16], OpenMP[17], PGI[18], HMPP[19] などプログラム上に指示行 (プラグマ) を挿入する事で, アクセラレータ上での並列化を可能にするものも盛んに研究され増えつつあるが, 筆者らの扱う大規模疎行列の線形問題や, 素粒子原子核における積分計算等においては, まだ満足のいくようなパフォーマンスは得られておらず, 現状においては, 最終的な手段としてアーキテクチャ固有の高速化についても自分で手の入れられる点の利点は大きい.

5. OpenCL を用いた一般化コードの構築

格子 QCD のシミュレーションは, オーダー 100 万 ×100 万の行列演算処理を伴うため, 計算機の発展と共にその道を歩んできたと行っても過言ではない. 近年においては,GPU を用いた大規模シミュレーション [24], [25], [26], [27] や OpenCL を用いた実装の試みなども行われている [23]. しかしながら, これらは高速化に重点を置いたコードであるため, 初心者にとっては, なかなか敷居が高いのも事実である. 例えば,OpenCL を用いたプログラムの実行の際の一般的な手順は以下のようになる.

(1) プラットフォームの特定
(2) デバイスの特定
(3) コンテキストの作成

(4) コマンドキューの作成
(5) メモリオブジェクトの作成
(6) カーネルファイルの読み込み
(7) プログラムオブジェクトの作成
(8) デバイスプログラムのビルド
(9) カーネルオブジェクトの作成
(10)カーネル引数の指定
(11)カーネルの実行（コマンドキューへの投入）
(12)メモリオブジェクトからの結果の読み込み
(13)オブジェクトの解放

これらの手続きは, 関数名 "cl "から始まる拡張 C 言語の関数で処理することになる. 利用者にとっては, ベースコードセットの C ++言語での記述に加え, 上記のような手続きのための, 関数を覚えなければならない. このような手続きが増えてしまうのは, 我々の重視する可読性, 可搬性のあるコードセットという理念もさることながら, デザインパターンとしても受け入れにくい (図 5 参照). そこで, 今回検討したのは,OpenCL 関数を広い意味でラップし, その手続きを媒介し, さらに, メモリやカーネルを管理するマネージャークラスである. このマネージャークラスは, シングルトンクラスとして定義してある. まず,main プログラムにおいて実行される初期化の段階で, コマンドキューの作成までを行い, それらを保持する.OpenCL 実行が可能なクラスでは, 事前にフラグで切り分け, 利用する場合は, マネージャーに対して, 自クラスが保持しているカーネルソースを指定し, カーネルのビルドを要求する. 要求を受けたマネージャークラスは, カーネルのビルドを行い, カーネルオブジェクトをビルドを要求したクラスに戻す. 対象クラスは, それを保持し, 自分が呼ばれるタイミングでそのカーネルを用いて, 今度はマネージャーに実行要求を必要な引数と共に渡す. この手続きは, 外見上は通常の関数呼び出しの何も変わらない. 実行が終了すれば, 結果が戻されるのも同様である. これらの実装によって, 既存のコードセットのほとんどを変えることなく OpenCL による実装を加える事が可能になる.

なお, 現在実行性能の検証や, 精度比較などを行っており, それらについては講演時に報告予定である.

6. 謝辞

本研究は科研費（20105001,20105005）及び計算科学連携拠点分野 5 の助成を受けたものである.

参考文献

[1] available from ⟨http://www.khronos.org/opencl/⟩
[2] David A. Patterson.: *The Trouble With Multicore*, IEEE Spectrum (2006).
[3] S. Motoki and A. Nakamura: *Development of QCD-code on a CELL Machine*, PoS(LATTICE 2007)040
[4] S. Motoki, Y. Nakagawa, K. Nagata, K.Hashimoto, K.Mizumaru and A. Nakamura: *Development of lattice QCD Tool Kit on cell broadband engine processor*, PoS(LAT2009)039
[5] Shinji Motoki, Atsushi Nakamura, Koichi Hashimoto, Kiyoshi Mizumaru: *Problem Solving Environment for Lattice QCD on Cell/B.E.*, Journ. Convergence Information Technology, Volume 5, Number 4, June 2010, pp. 187 194
[6] Shinji Motoki and Atsushi Nakamura: *Problem Solving Environment of Lattice QCD*, Proceeding of 5th International Conference on Computer Sciences and Convergence Information Technology, pp249-252 ed. by F. Ko, S. Kawata, S. Fong and Y. Na, AICIT, IEEE Conference Record number #17832
[7] S. Motoki and A. Nakamura: *Lattice gauge theory on a multi-core processor, Cell/B.E.* Procedia Computer Science, Vol.4, pp.860-868, (2011).
[8] G. Shi, V. Kindratenko and S. Gottlieb, The XXVI International Symposium on Lattice Field Theory, PoS (LATTICE2008) 026, 2008; Int. J. Parallel Prog. 37, pp.488-507, 2009.
[9] F. Belletti et al., *QCD on the Cell Broadband Engine*, The XXV International Symposium on Lattice Field Theory, PoS (LATTICE 2007) 039, 2007.
[10] G. Shi et al., Application Acceleration with the Cell Broadband Engine Computing in Science & Engineering, Vol.12, Issue:1, pp.76-81, 2010.
[11] H. Baier et al., The XXVII International Symposium on Lattice Field Theory, PoS (LAT2009) 001, 2009.
[12] available from ⟨http://physics.indiana.edu/ sg/milc.html⟩
[13] available from ⟨http://qcdoc.phys.columbia.edu/cps.html⟩
[14] available from ⟨http://usqcd.jlab.org/usqcd-docs/chroma/⟩
[15] available from ⟨http://nio-mon.riise.hiroshima-u.ac.jp/ LTK⟩
[16] available from ⟨http://www.openacc-standard.org/⟩
[17] available from ⟨http://openmp.org/wp/⟩
[18] available from ⟨http://www.pgroup.com/⟩
[19] available from ⟨http://competencecenter.hmpp.org/⟩
[20] M. A. Clark, R. Babich, K. Barros, R. C. Brower, C. Rebbi, *Solving Lattice QCD systems of equations using mixed precision solvers on GPUs*, arXiv:0911.3191, Computer Physics .Communications.Vol.181, pp.1517-1528, 2010.
[21] J. R. Rice,R. F. Boisvert, *From scientific software libraries to problem-solving environments* , Computational Science & Engineering, Vol.3, Issue:3, pp.44-53, 1996.
[22] K. Wilson, Phys. "Confinement of quarks", Rev. D10, 2445, 1974.
[23] Matthias Bach, Owe Philipsen, Christopher Pinke, Christian Schfer, Lars Zeidlewicz arXiv:1112.5280v1, Proceedings of the XXIX International Symposium on Lattice Field Theory, PoS LATTICE2011 (2011) 044.
[24] C. Bonati, G. Cossu, M. D'Elia and P. Incardona, arXiv:1106.5673 [hep-lat].
[25] R. Babich, M. A. Clark and B. Joo, arXiv:1011.0024 [hep-lat].
[26] M. A. Clark, R. Babich, K. Barros, R. C. Brower and C. Rebbi, Comput. Phys. Commun. **181** (2010) 1517 [arXiv:0911.3191 [hep-lat]].
[27] R. Babich, M. A. Clark, B. Joo, G. Shi, R. C. Brower and S. Gottlieb, arXiv:1109.2935 [hep-lat].
[28] available from ⟨http://suchix.kek.jp/bridge/Lattice-code/⟩