Vol.2013-HPC-141 No.12
2013/10/1

# 量子モンテカルロ法による物性スペクトル計算への多倍長演算の適用

濱口 信行[1,a)]　石川 正[1,b)]　岩野薫[1,c)]

**概要**：物性スペクトル決定に量子モンテカルロ法を適用した場合、電子グリーン関数は系のサイト数を次元とする行列の行列積として表されるが、それを数値的に求めようとすると 4 倍精度演算でも計算が困難なパラメータ領域が存在していた。本報告では、まず、グリーン関数を多倍長演算を適用して十分精度良く計算すること、そしてさらに並列用多倍長演算ソースの作成により十分高速に計算することに成功し、計算出来るパラメター領域を大きく拡大することが出来た。

# Application of multi-precision operation to material spectrum simulation with quantum Monte Carlo method

## 1. はじめに

固体中の電子が強いクーロン斥力で相互作用をした系 は摂動論などの近似法が使えず、いわゆる非摂動論的な方 法か必要とされる。中でも量子モンテカルロ法は原理的には扱える電子系のシステムサイズ、次元性、相互作用の強さに限界が無く、今後の手法の発展に大きな可能性を秘めている。

本研究では 特に電子系の光学スヘクトルに関心があり、量子モンテカ ルロ法によっていわゆる電子虚時間グリーン関数、$G(\tau)$ を計算 し、それからスヘクトル、$S(\omega)$ を求めること を考えている **?**。この両者は、

$$G(\tau) = -\int_{-\infty}^{\infty} dt \frac{e^{-\tau\omega}}{1+e^{-\beta\omega}} S(\omega) \qquad (1)$$

という関係にあり、量子モンテカルロ法によってまず求まるのは左辺の $G(\tau)$ である。従って、それから $S(\omega)$ を求めるためには 1 種の逆ラプラス変換を行う必要があるが、数値的にしか得られていない前者に対しては解析的に出来ない。そのため、Pade 近似、最小二乗法、最大エントロピー法（MEM）などの方法が試みられてきたが、Pade 近似は仮定する関数形に柔軟性がないこと、最小二乗法はインプットの $G(\tau)$ が精度良く求まっていないこと、MEM は任意性が入りやすいこと、などの理由により不満足な結果になる例が多かった。

しかしながら、中には有望な示唆を与える研究も報告されており、例えば、富田等は最小二乗法を用いる際、４倍精度演算を適用し出来るだけ精度良く $G(\tau)$ を計算することを初めて提案した [2]。確かに彼らの方法は後述するようにあるモデルパラメータ範囲では有効であり、それは十分価値ある知見であったが、しかし、その範囲を超えた場合計算は極めて困難となっていた。また、松枝等は MEM を用いてスペクトル計算を行った。彼らは量子モンテカルロ法と同時にいわゆる動的密度行列繰り込み群法（DDMRG 法）も実施したが、前者は有限温度、後者は絶対零度に限られていたものの、前者の最低温スペクトルが後者のスペクトルに良くつながるように見えるなど非常に良質の結果を与えた [3]。とはいえ、彼らの結果も同様なパラメター範囲に限定されているという点では富田等と同じであった。我々は、富田等と同じく出来るだけ精度良く $G(\tau)$ を計算するという方針で、しかも、なるべくなら人為性の入り易い MEM を避けて、最小二乗法だけでスペクトルを計算することを目的として、以下で説明するような多倍長精度計算を試みた。

使用した計算機は性能測定は SR16000/M1　1 ノード (物理コア 32、論理コア 64、理論演算性能 980.48GFLOPs.) で行い、精度検証には、SR16000/M1 及び別アーキテク

[1] 高エネルギー加速器研究機構
茨城県つくば市大穂 1–1
[a)] hamagu@post.kek.jp
[b)] tadashi.ishikawa@kek.jp
[c)] kaoru.iwano@kek.jp

チャーの intel サーバー x5570(周波数 2.93GHz) を使用している。

## 2. 計算概要と必要とするビット数

まず、量子モンテカルロ計算では 1 つのパス (配位)、$x \equiv [S_i^m]$ に対し、グリーン関数 $G(\tau, x)$ を求めその統計平均 $G(\tau)$ を求める事になる。ここで、$i$ は 1 から $N$ までの値を取り、対象とする格子系の実空間サイトに付けた番号であり、全サイト数を $N$ とする。一方、$L$ は温度 $T$ の逆数 $\beta = 1/(k_B T)$ ($k_B$ はボルツマン定数) の分割数であり、$m$ はその分割点に対応し 0 から $L$ までの値を取る。

与えるパラメターは、いわゆる 1 次元ハバード模型において電子のホッピングを規定する $t_0$、および、電子間クーロン斥力を規定する $U$ であり、サンプリングにより $\pm 1$ の値を取るパス $[S_i^m]$ を定め、サイズ $N$ の行列を $L$ 個 $(B_1, B_2, ....., B_L)$ を求めて、これから以下のようなサイズ $N$ の行列を $L+1$ 個定める。

$$\mathbf{G}_m = -(B_m B_{m-1} \dots B_1 B_0)[I + B_L B_{L-1} \dots B_1]^{-1} \quad (2)$$

ここで、$m = 0, 1, 2 \dots L$,$B_0 = I$ であり、グリーン関数は行列 $\mathbf{G}_m$ のトレースであり、一つのパスに対して次式のように $L+1$ 個の値を求める事になる [4][5]:

$$G(\tau = m\Delta) = \mathrm{Tr}[\mathbf{G}_m] \quad (3)$$

ただし、ここで $\Delta = \beta/L$ と定義する。

この計算では、$B_m B_{m-1}.....B_1$ の行列積をそのまま計算すると桁落ちが大きくなるので、以下の QDR 分解を用いている。QDR 分解とは、行列 V をグラムシュミットの直交化で V=QR(Q:直交行列、R:上三角行列) とし、R の対角要素で同じ行の列要素を割る。R の対角要素をベクトル D として、R を対角要素 1 の上三角行列にして V=QDR とする。[6][7][8]: QDR 分解により、$B_m B_{m-1}.....B_1$ を

$$\begin{aligned} B_1 &= Q_1 D_1 R_1 \\ B_2 Q_1 &= Q_2 D_2 R_2 \\ B_2 B_1 &= Q_2 D_2 D_1' R_2' R_1 = Q_2 D_2 \text{"} R_2 \text{"} \\ &\vdots \\ B_m Q_{m-1} &= Q_m D_m \text{"} R_m \text{"} \\ B_m B_{m-1}.....B_1 &= QDR \end{aligned} \quad (4)$$

という手順で計算する。

### 2.1 必要とするビット数

今回扱ったケースは、静的かつ規則的なパス $S_i^m = (-1)^i$ に対するものなので解析が容易で、計算結果の検証は $\mathrm{Tr}(G_L) = -1$ という条件で確認することができる。また、$t_0 = 1$、$N = 100$、$L = 448$ と固定し、計算でのパラメータは $\beta$、$U$ の 2 つとしている。$y = \sqrt{\beta U L}$ とし、$P = e^y$、E(m)=|Tr($G_m$)| とすると以下の様になる。(1) プログラム実行中に現れる数値の最大値は $P^2$、
0 以外の絶対値最小値は $\frac{1}{P^2}$。
(2) 行列やベクトル要素に現れる数値の最大値は $P$、
0 以外の絶対値最小値は $\frac{1}{P}$。
E(0)=1、E(L)=1、m=$\frac{L}{2}$ で E($\frac{L}{2}$)=$\frac{1}{\sqrt{P}}$ で最小となる。
よって、条件数は $\sqrt{P}$ となり、演算に必要なビット数は $\log_2(\sqrt{P}) = \frac{1}{2}\log_2 P$ となる。

今回 4 倍精度で正しく計算できなかったケースは、$\beta$=10、$U$=5 で、演算に必要なビット数は 107.96 である。有効ビット数は、計算機のアーキテクチャーにより異なるが、SR16000/M1 では 106 ビット、x5570 で 113 ビットであった。Tr($G_L$) の実測結果は以下の通りである。

- 行列積（(2) 式）をそのまま計算した場合
  -0.1754358602D+01
- QDR 分解（(4) 式）を SR16000/M1 で計算した場合
  -0.1005312500Q+01
- QDR 分解（(4) 式）を x5570 で計算した場合
  -0.1000000471D+01

これらの結果より、QDR 分解による精度改善が見て取れる。また、必要なビット数と有効ビット数の関係が結果の精度とよく対応している。

## 3. 多倍長演算の最適化と並列化

多倍長演算では、2 つの方式がある。
(1) 多倍長変数を複数個 ($m > 2$) の倍精度変数の和で表す。
(2) IEEE754-2008 の 4 倍精度変数の仮数部のビット数を多くする。
( P 倍精度変数の仮数部のビット数=P×32 − 16)

一般的には、性能は 8 倍精度までは、(1) の方式が良く、精度的には (2) の方式が良い。　本研究では (1) の方式が中心で、精度の詳細部分やアーキテクチャーの違いによる比較に補助的に (2) の方式を x5570 で行った。

多倍長変数を複数 ($m > 2$) の倍精度変数の和で表す方式の場合の注意点は以下の通りである。
(1) 演算順序の保証と、複数の倍精度変数は絶対値の大きい順に並べる必要がある。
(2) 乗算においては、乗加算命令のある機種とない機種があり、複数のアーキテクチャーの計算機で動作させるには乗加算命令のないソースに統一する必要がある。
(3) 演算量は、加減算で倍精度変数の個数 $m$ の二乗に比例し、乗算、除算、平方根演算では $m$ の三乗に比例する [9]。

これらの事と最適化及び並列化との関連は以下の様になる。
(1) では演算順序の保証する条件下の最適化オプションの選択が必要である。
(2) ではソースのメンテナンスを考慮して、基本演算 (加減算、乗算、平方根) はサブルーチン化するのが良く、サブルーチンコールのオーバーヘッドを削減するオプションの選択が必要である。
(3) では演算量が大きくなると、最適化と並列化に影響がでるため、影響の出ないオプションの選択が必要である。

この問題では行列の次元数 $N = 100$ と小さいため、演算量が $N$ の三乗になる行列積計算、QDR(Q、R 行列、D ベクトル) 積計算部分をサブルーチン化して並列化する。
性能測定結果は以下の値になり並列化効果が良くでている。ここで 32smp とは 1 ノード 32 物理コア (smt=off) での実行を示しており、64smp とは 1 ノード 64 論理コア (smt=on) の実行を示している。

**表 1　SR16000/M1 多倍長行列積計算、$N = 100$、実行回数 1000 回、実行時間 (秒)**

| 演算精度 | single 実行 | 32smp | 64smp | 最大並列化効果 |
|---|---|---|---|---|
| 4 倍精度 | 15.786 | 0.642 | 0.481 | 32.8 |
| 6 倍精度 | 54.099 | 2.204 | 1.475 | 36.7 |
| 8 倍精度 | 92.124 | 3.797 | 2.420 | 38.1 |
| 10 倍精度 | 173.321 | 6.964 | 4.823 | 35.9 |

## 4. 6 倍精度、8 倍精度演算での実行結果

### 4.1 精度検証結果

$\beta$、$U$ の値に対し必要なビット数と実測結果は表 2 の様になった。ここで OK は 10 進 10 桁以上一致、NG は最初の 2 桁以下のみ一致　(全桁不一致も含む)、を示していて、それ以外は実測値を示した。
有効ビット数は 6 倍精度が 159 ビット、8 倍精度が 212 ビットで必要とするビット数と有効ビット数の関係と実測結果には良い一致が見られる。また DATA1、DATA2、DATA3 の 3 ケースの値は以下の通りである。

6 倍精度
DATA1 $\beta$=10、$U$=10 -0.9999999857Q+00
DATA2 $\beta$=20、$U$=5 -0.9999987411Q+00
8 倍精度
DATA3 $\beta$=20、$U$=9 -0.999998945Q+00

**表 2　必要なビットと実測結果**

| $U$ | $\beta$ | ビット数 | 6 倍精度 | 8 倍精度 |
|---|---|---|---|---|
| 5 | 10 | 108 | OK | OK |
| 6 | 10 | 118 | OK | OK |
| 7 | 10 | 128 | OK | OK |
| 8 | 10 | 137 | OK | OK |
| 9 | 10 | 145 | OK | OK |
| 10 | 10 | 153 | DATA1 | OK |
| 5 | 20 | 153 | DATA2 | OK |
| 6 | 20 | 167 | NG | OK |
| 7 | 20 | 181 | NG | OK |
| 8 | 20 | 193 | NG | OK |
| 9 | 20 | 205 | NG | DATA3 |
| 10 | 20 | 216 | NG | NG |

### 4.2 4 倍精度演算と 6 倍精度、8 倍精度演算の差異

変数を複数個 ($m \geq 2$) の倍精度変数の和で表す場合、$m = 2$ か $m > 3$ では異なる処理が必要となる。$m = 2$( 4 倍精度演算) では、演算後に得られた結果 a=$a_1$+$a_2$ (a:4 倍精度変数、$a_1$、$a_2$： 倍精度変数) では自動的に絶対値の大きい順に並ぶが、$m > 3$ では別の処理を施さないと保証されない。
ここでは、絶対値の大きい順に並ぶ事が保証されない演算を非正規化演算とする。

8 倍精度演算では、8 倍精度変数を 4 つの倍精度変数の和で表す以外にも、2 つの 4 倍精度変数の和で表す方式も考えられる。これは作成するソースプログラムの数 (ステップ数) の観点から見ると後者では、およそ $\frac{1}{10}$ となる。

ただし精度からみると、IEEE754-2008 形式では、有効ビット数が 212 ビットから 226 ビットと増え、表現可能な数値範囲も大きく広がるが、SR16000/M1 では、表現可能な数値範囲は変わらず、加減算は非正規化 8 倍精度演算となり、乗算は最悪ケース非正規化 6 倍精度演算まで精度が低下する事がある。理由は以下の通りである。
SR16000/M1 で 8 倍精度変数が以下の様に表されるとする [10]。

a=$a_0$+$a_1$+$a_2$+$a_3$=$c_0$+$c_1$、$c_0$=$a_0$+$a_1$、$c_1$=$a_2$+$a_3$
b=$b_0$+$b_1$+$b_2$+$b_3$=$d_0$+$d_1$、$d_0$=$b_0$+$b_1$、$d_1$=$b_2$+$b_3$
ここで $a_i$、$b_i$ は倍精度変数、$c_i$、$d_i$ は 4 倍精度変数とする。

例えば加減算の結果が $c_0$+$c_1$ なった場合は $c_0$ の絶対値は $c_1$ の絶対値以上ではあるが $a_1$ と $a_2$、$a_3$ の絶対値に関しては大小関係は定まらない。

乗算では、

$$\begin{aligned} a \times b = & a_0b_0 + (a_0b_1 + a_1b_0) \\ & +(a_0b_2 + a_1b_1 + a_2b_0) \\ & +(a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0) \\ & +(a_1b_3 + a_2b_2 + a_3b_1) \end{aligned}$$

に対し、

$$\begin{aligned} a \times b = & c_0d_0 + c_0d_1 + c_1d_0 \\ = & (a_0b_0 + a_0b_1 + a_1b_0 + a_1b_1) \\ & +(a_0b_2 + a_0b_3 + a_1b_2 + a_1b_3) \\ & +(a_2b_0 + a_2b_1 + a_3b_0 + a_3b_1) \end{aligned}$$

と $a_2b_2$ の項が含まれない。$a_3 = b_3 = 0$ の場合、後者では
$a \times b$=$a_0b_0 + a_0b_1 + a_1b_0 + a_0b_2 + a_1b_1 + a_2b_0 + a_1b_2 + a_2b_1$

となり、非正規化 6 倍精度演算の乗算結果と一致する事になる。

8 倍精度変数を 2 つの 4 倍精度変数の和として作成ソース量が削減できて、結果も正しいものが得られたケースに、rump's の例題がある [11]。
a=77617.0、b=33096.0
f=$333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}$
の計算で理論解は $-\frac{54767}{66192}$=-0.827396059946
この計算では有効ビット数が 121 ビット以上必要なため、4 倍精度演算では 1.172603940053 という結果となっている。また非正規化演算の影響を受けないため、2 つの 4 倍精度変数の和で表す方式でも正しい結果が得られた。

量子モンテカルロ法による物性スペクトル計算では
β=10、U=5 に対しては、全て除算、平方根計算は 2 つの 4 倍精度変数の和の形式とし

加減算、乗算は 2 つの 4 倍精度変数の和の形式の結果
　-0.1000328780Q+01
加減算、乗算は 4 つの倍精度変数の和の形式の結果
　-0.1000000000Q+01
加減算は 4 つの倍精度変数の和の形式の結果
　-0.1000378650Q+01
乗算は 4 つの倍精度変数の和の形式の結果
　-0.1000517641Q+01

となっている。すなわち、加減算、 乗算のどちらか一方、または両方を 2 つの 4 倍精度変数の和の形式にするとほぼ 4 倍精度演算より 10 進 1 桁だけ精度はよくなるが、絶対値の大きい順に並ぶ事を保証した 6 倍精度演算の精度以下となっている。
加減算、乗算をともに 4 つの倍精度変数の和の形式にすると、以下の様に絶対値の大きい順に並ぶ事を保証した 8 倍精度演算の結果と一致する。

β=10、U=10
　-0.1000000000Q+01
β=20、U=6、7、8
　-0.1000000000Q+01

以上から 量子モンテカルロ法による物性スペクトル計算では 2 つの 4 倍精度変数の和の形式にして作成するソースプログラムのステップ数を削減する事は出来ない事が判明した。このため性能測定はすべて 4 つの倍精度変数の和の形式をとって行っている。

### 4.3 性能測定結果

性能評価の基礎データとしてβ=10、U=5 の 4 倍精度演算自動並列で測定した時の実行時間は以下の様になっている。
single 実行　65.989 秒、32smp 実行　9.327 秒、64smp 実行 11.04 秒
表 1 の N=100 の行列積の計算では、single 実行時 15.786 秒に対し、64smp で 0.481 秒と 32 倍以上の並列化効果が出ている事から、演算量 N の一乗、二乗の計算部分の全体の実行時間に対する実行時間比率が大きくなり、プログラム全体での並列化効果を小さくしている事がわかる。

性能測定とチューニングは、基本演算（加減算、乗算、除算、平方根演算）をサブルーチン化して single 実行する事から行った。
実行パラメータはそれぞれの精度の特徴から
6 倍精度　β=10、U=6 (4 倍精度では正しく計算出来なかった)
8 倍精度　β=20、U=8 (結果が 10 進 10 桁一致する最大数)
に設定している。
性能チューニングとその効果を順を追って示す。

#### 4.3.1 サブルーチンコールオーバーヘッド削減オプションの効果

サブルーチンコールオーバーヘッド削減なし
　6 倍精度　612 秒　　8 倍精度　1190 秒
サブルーチンコールオーバーヘッド削減あり
　(コンパイルオプションのみ)
　6 倍精度　413 秒　　8 倍精度　695 秒
となり、約 40% の実行時間の削減が得られた。

#### 4.3.2 行列積計算、QDR 積計算の並列化効果

行列積計算、QDR 積計算のサブルーチン化（並列化）
6 倍精度　32smp 105 秒 64smp 112 秒
8 倍精度　32smp 174 秒 64smp 192 秒
約 4 倍の性能向上が得られた。

#### 4.3.3 IF 文削除によるサブルーチンコールオーバーヘッド削減効果

多倍長演算での変数は、複数の倍精度変数の和で表されるが、複数の倍精度変数を絶対値の大きい順に並べる必要が有り、ＩＦ文を使用してその操作を行っている [10]。

これはサブルーチンコールオーバーヘッド削減効果を阻害する事がある。このため、倍精度変数の加算の結果を 2 つの倍精度変数の和 (絶対値の大きい順に) で表す方法を繰り返す事により IF 文を削除した。

a、b、c、d:倍精度変数　a+b=c+d の計算
c=a+b
t=c-a
d=(a-(c-t))+(b-t)

このチューニングにより、
サブルーチンコールオーバーヘッド削減あり
(ソース修正を追加)
6 倍精度　32smp 28.837 秒 64smp 23.743 秒
8 倍精度　32smp 51.739 秒 64smp 42.074 秒
と 更に 4 倍の性能向上が得られた。

#### 4.3.4 性能向上まとめ

4 倍精度演算と比較し、実行時間は 6 倍精度で 2.5 倍、8 倍精度で 4.5 倍で演算量での 3 倍、7 倍より短い時間で終了している [9]。

## 5. 10 倍精度演算

### 5.1 10 倍精度演算使用の背景

6 倍精度、8 倍精度のところで、β=20、U=9、U=10 では必要ビット数がそれぞれ 205、216 ビットで 8 倍精精度の有効ビット数 212 では計算が困難になっていた。
β=20、U=9　-0.9999998945Q+00
β=20、U=10　-0.1086500000Q+02

IEEE754-2008 形式の 8 倍精度では有効ビット数 241 のため結果は 10 進 10 桁まで一致している。
β=20、U=9　-0.1000000000D+01
β=20、U=10　-0.1000000000D+01

IEEE754-2008 形式の 2 つの 4 倍精度変数の和で有効ビット数 226 の擬似 8 倍精度演算を行うと次のようになる。
β=20、U=9　-0.1000000000D+01
β=20、U=10　-0.1000000011D+01

ここでも必要とするビット数と有効ビット数の関係と実測結果の良い一致が見られている。
このため、IEEE754-2008 8 倍精度演算で正しい結果の得られたパラメータ領域よりさらに広い領域を計算するために 10 倍精度演算を使用する。

### 5.2 10 倍精度演算特有の問題点

10 倍精度演算を使用した際、次ぎの問題が発生した。
(1) プログラム実行中に表現可能な数値範囲を超す場合があり、演算中にオーバーフローが発生する。
(2) 多倍長変数を複数の倍精度変数の和で表す場合、大きな数の逆数の精度が悪くなる。

U=10 を固定した場合、β の値に対するプログラム実行中に現れる最大数 $10^n$ と精度の良い結果を得るのに必要なビット数 P は表 3 の様になる。

表 3　プログラム中に現れる最大数 $10^n$ と必要なビット数

| $\beta$ | $n$ | $P$ |
|---|---|---|
| 20 | 260 | 216 |
| 21 | 266 | 221 |
| 22 | 273 | 227 |
| 23 | 279 | 232 |
| 24 | 285 | 237 |
| 25 | 291 | 241 |
| 26 | 296 | 246 |
| 27 | 302 | 251 |
| 28 | 308 | 256 |
| 29 | 313 | 260 |
| 30 | 318 | 265 |

この表から (1) に関しては 10 倍精度演算の有効ビット数は 265 ビットで、必要ビット数から見ると U=10、β=28 は計算可能だが、
プログラム実行中にオーバーフローの発生が推測される。
この問題は IEEE754-2008 形式では発生しない。
(2) に関しては、大きな数 $10^n$ の逆数 $10^{-n}$ の持つ有効ビット数は演算の有効ビット数を $m$ とすると、$min(m, (1074 - n/\log_{10}(2)))$ となる.
これは、平方根計算に影響を与える。

例えば、U=10、β=21、22 では必要ビット数はそれぞれ 221、227 ビットのため計算可能。
通常多倍長演算での平方根計算は除算を避けるため、逆数の平方根を求める方式がとられるので [10]、有効ビット数分の精度が保持できなくなり、精度の悪い結果を得る可能

性が推測される。

実際これまでの平方根計算アリゴリズムを使用して 10 倍精度演算で実行した結果は以下の様になっている。

β=20、U=10　-0.1000000000Q+01
β=21、U=10　-0.9999988856Q+00
β=22、U=10　-0.8538546954Q+03

これに対し IEEE754-2008 形式の 8 倍精度演算結果は

β=20、U=10　-0.1000000000D+01
β=21、U=10　-0.1000000000D+01
β=22、U=10　-0.9999999996D+00

となり、10 倍精度より有効ビット数が少ない 241 ビットにもかかわらず結果の精度は良くなっている。

### 5.3 平方根計算アルゴリズムの変更

多倍長計算での平方根計算には以下の方法がある。

(1) 逆数平方根算出方式
$x_0 = \frac{1}{a}$ (倍精度)
$x_{n+1} = x_n + \frac{x_n(1-ax_n^2)}{2}$
$\sqrt{a} = a \times x_{last}$

(2) 除算使用方式
$x_0 = \sqrt{a}$ (倍精度)
$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
$\sqrt{a} = x_{last}$

(3) 除算使用方式（最終ビット調整)
$x_0 = \sqrt{a}$ (倍精度)
$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
最終反復時は $x_{n+1} = \frac{1}{2}(x_n - \frac{a}{x_n}) + \frac{a}{x_n}$
$\sqrt{a} = x_{last}$

(1)(2)(3) をそれぞれ方式 1、方式 2、方式 3 として引数 P=$10^n$ とし、上記 3 方式で求めた結果 $\sqrt{P} = 10^{\frac{n}{2}}$ と 10 を $\frac{n}{2}$ 回掛けた値と一致するビット数を求める事により結果の検証を行なった (表 4)。

**表 4　平方根計算精度検証結果 (一致するビット数)**

| n | 方式 1 | 方式 2 | 方式 3 |
|---|---|---|---|
| 240 | 261 | 262 | 262 |
| 250 | 241 | 260 | 261 |
| 260 | 207 | 261 | 261 |
| 270 | 184 | 257 | 258 |
| 280 | 143 | 260 | 260 |

U=10、β=21 から 27 では、プログラム実行時に平方根の引数は $10^{265}$ から $10^{300}$ の大きさとなり、このテスト結果とプログラムの結果の精度の対応が良くとれている。

平方根計算のアルゴリズムを方式 2 とすると、
U=10、β=21、22、23、24、25、26 OK
U=10、β=27　-0.9999999998Q+00
方式 3 とすると U=10、β=27 で OK となる。

β=28、29、30 では推定どうりオーバーフローが発生している。尚、これらのパラメータ領域では表現できる数値範囲の制限を受けない IEEE754-2008 形式の 4 倍精度変数の 3 つの和で表した擬似 12 倍精度演算ではβ=40、U=10 で、4 つの和で表した擬似 16 倍精度演算、5 つの和で表した擬似 20 倍精度演算ではβ=50、U=10 で 10 進 10 桁一致する結果が得られている。

### 5.4 性能測定結果

データはβ=20、U=10 で実施した。最初は 32smp　649 秒、64smp　714 秒とかかっている。
原因は QDR 計算のサブルーチンで演算量が多くなった事により最適化が阻害され、並列化が行われなかった事による。
QDR 計算を行列ベクトル積と行列積計算にわけ、並列化して実行した結果は、
32smp　396 秒、64smp　414 秒と改善されたが、4 倍精度演算の実行時間の 42 倍と性能はまだ不十分であった。
このため、QDR 計算のサブルーチンは元のソースを使用し、
コンパイラの最適化に使用するワークエリアサイズの制限を外すオプションで最適化、並列化を行う事を検討した。

**表 5　10 倍精度演算、N=100、実行回数 1000 回、QDR 計算の実行時間 (秒)**

| スレッド数 | 制限あり | 制限なし |
|---|---|---|
| 1 | 295.463 | 269.01 |
| 32 | 295.750 | 10.807 |
| 64 | 322.055 | 7.416 |

コンパイラの最適化に使用するワークエリアサイズの制限に関する性能測定結果は表 5 の様になった。

ワークエリアサイズの制限を外すオプションで最適化、並列化を行い
32smp　110 秒、64smp　91 秒
という結果が得られ、4 倍精度演算での実行時間の 10 倍未満の時間で計算が出来た。
演算量は 4 倍精度演算の 12.5 倍 [9] であり、10 倍精度演算

においても演算量比より短い時間で終了している。

## 6. まとめ

4 倍精度演算で β=10、U=5 というパラメータでの結果の精度が良くなかったが、10 倍精度演算を用いると β=27、U=10 まで正しく計算できるようになり、パラメータ領域を拡大出来た. 実行時間も 10 倍精度演算で 4 倍精度演算での１０倍未満と演算量比率以下の時間で実行が可能になった。
また、表現できる数値範囲の制限により、複数の倍精度変数の和として表す多倍長演算には計算可能なパラメータ領域に制限が加わる事が示された。

今後の課題としては、
(1) GPU 等のアクセラレータへの 10 倍精度演算の適用。
(2) IEEE754-2008 の 4 倍精度のデータ形式を拡張した多倍長計算で、より条件の厳しいパラメータ領域での評価。
(3) 6、8、10 倍精度演算ソースを real*8 から real*16、乗算の定数 r=$2^{27}+1$ を r=$2^{57}+1$ とする事で IEEE754-2008 の 4 倍精度のデータ形式を拡張した擬似 12 倍精度演算、擬似 16 倍精度演算、 擬似 20 倍精度演算ソースの並列化により、より条件の厳しいパラメータ領域での実行。
(4) IEEE754-2008 の 4 倍精度のデータ形式を持つアクセラ レータへの適用。
の検討がある。

これらの情報の更新は 3 か月毎に HPCI 戦略分野 5 の高性能計算の扉 [12] で行っている。

**謝辞** 本研究は HPCI 戦略プログラム分野 5「物質と宇宙の起源と構造」、JSPS 科研費 23540328 の助成を受けたものである.

### 参考文献

[1] 第 3 回多倍長計算フォーラム量子モンテカルロ法における物性スペクトル計算 岩野薫 KEK 物構研 http://suchix.kek.jp/mpcomp/20130308-forum/slide-ki.pdf
[2] N. Tomita and K. Nasu、 Phys. Rev. **B** 3779、 (1997).
[3] H. Matsueda et al.、 Phys. Rev. **B** 72、 075136 (2005).
[4] 偏微分方程式と境界値問題：田辺行人、中村宏樹 東大出版会 1993 年
[5] 偏微分方程式とフーリエ解析：中村宏樹 東大出版会 1993 年
[6] 数値解析 森正武 共立出版 2002 年
[7] スーパーコンピュータとプログラミング 島崎真昭 共立出版 1989 年
[8] 行列計算ソフトウエア WS、スーパーコン、並列計算機 小国力、村田健郎、三好俊郎、Dongarra、jack、J 長谷川秀彦 丸善株式会社 1991 年
[9] Bailey アルゴリズムによる多倍長演算の性能評価情報処理学会研究報告 2008-HPC-114-5
[10] Quad-Double Arithmetic Algorithms、Implementation、and Application Yozo Hida Xiaoye S.Li David H.Bailey October 30、2000
[11] A review on Interval Computation Software and Applications Chenyi Hu、shanying Xu、and Xiaoguang Yang http://happy.dt.uh.edu/hu/Papers/intCRV.pdf
[12] http://www.jicfus.jp/field5/jp/promotion