Vol.2012-ARC-202 No.23
Vol.2012-HPC-137 No.23
2012/12/14

# 4倍精度基本線形代数ルーチン群QPBLASの紹介とアプリケーションへの応用

山田 進[1,3,a)]　佐々 成正[1,b)]　今村 俊幸[2,3]　町田 昌彦[1,3]

**概要**：計算機を用いた演算では，無限桁数の実数を有限桁で表現するための丸め誤差や，演算の際に桁落ちや積み残し等が発生するが，これまでの計算機の規模のシミュレーションでは倍精度演算を利用することで，これらの影響はあまり問題にならなかった．しかし，「京」コンピュータのような超大規模な計算機の性能を全て利用するような超大規模シミュレーションでは演算量が多くなるため，これらの影響によりシミュレーション結果の精度に問題が生じる可能性がある．そこで，我々は基本線形演算ライブラリ BLAS を Bailey の double-double アルゴリズムを利用して 4 倍精度化し QPBLAS として公開している．本発表では，QPBLAS および QPBLAS を利用して開発した 4 倍精度固有値計算ライブラリ QPEigenK を紹介し，その有効性や性能等を報告する．

## 1. はじめに

2012 年 9 月より 10PFLOPS 超の計算性能を誇る「京」コンピュータの一般利用が開始されたように，現在では超大規模の計算機を一般のユーザが利用できるような環境になりつつある．そして，このような大規模な計算機の性能を有効に利用するため，シミュレーションモデルも大規模化し，その計算量も増大している．計算機による演算では無限桁の実数を有限で表現して演算を行うため，演算 1 回ごとに誤差が混入するが，これまでのシミュレーションではあまり問題にはならなかったが，上記のような超大規模計算機の性能を有効に利用するようなシミュレーションを実施した場合，計算回数がこれまでよりも多くなり，誤差による影響が大きくなる可能性がある．そのため，得られた計算結果の有効精度がほとんど無いということになることも考えられる．実際，我々は地球シミュレータを利用して 375,000 次元の実対称行列の全固有値・固有ベクトルを計算したところ [1]，その精度は数桁であった．そのため，このような計算機の大規模化が進む現状では，この精度の問題を解決する必要がある．

この問題を解決する手段の 1 つに，計算で利用する計算精度をあげる方法がある．実際，現在の多くの Fortran コンパイラでは 4 倍精度実数変数である `real*16` をサポートしており，これを用いることで容易に高精度計算が可能になる．しかし，その計算速度は実数演算と比較し非常に多くの計算時間がかかるため実用的ではない [2]．そのため，倍精度変数等を組み合わせて高精度の計算を実現する様々な計算方法が提案されており，そのような計算方法を利用したライブラリが数多く公開されている（一例として [3]）．その 1 つに Devid H. Bailey が提案した二つの倍精度変数を組み合わせて 4 倍精度変数を表現する double-double アルゴリズムがある [4]．この方法は倍精度実数 (`real*8`) の演算を組み合わせることで 4 倍精度計算を実現する方法である．実際，これまでにも多くの double-double アルゴリズムを利用したライブラリ等が開発・公開されている [4]．さらに，Krylov 部分空間法に用いることで条件数の大きい行列を係数に持つ線形方程式を安定に計算することに成功するなど [2]，実用的なアルゴリズムである．そこで原子力機構では，この double-double アルゴリズムを利用した 4 倍精度 BLAS(QPBLAS) を開発し，また，これらを利用した 4 倍精度固有値計算ルーチン (QPEigenK) の開発を行っている．

以下に本論分の構成を記す．第 2 章では double-double アルゴリズムの簡単な紹介およびこれを用いて開発した 4 倍精度 BLAS の説明を行う．第 3 章では，これらを利用して開発した 4 倍精度固有値計算ソルバに付いて説明し，その性能を報告する．第 4 章はまとめである．

[1] 日本原子力研究開発機構 システム計算科学センター
JAEA CCSE, Kashiwanoha, Kashiwa, Chiba 277–8587, Japan
[2] 理化学研究所 計算科学研究機構
RIEKN AICS
[3] CREST(JST)
[a)] yamada.susumu@jaea.go.jp
[b)] sasa.narimasa@jaea.go.jp

## 2. 4倍精度演算

### 2.1 Bailey の double-double アルゴリズム

Bailey の double-double アルゴリズムでは，4倍精度浮動小数点変数 $a$ を二つの倍精変数 $a.hi$(上位データ) および $a.lo$(下位データ) を用いて $a = a.hi + a.lo$ と表現し，4倍精度演算を倍精度変数の演算で実現する方法である．このとき，$a.hi$ および $a.lo$ は通常の倍精度浮動小数点数であるため仮数部の精度は 52bit であり，2つの変数を利用することで 104bit の精度で表現できる．しかし，`real*16` の場合は仮数部を 112bit で表現している．そのため，double-double アルゴリズムは `real*16` と比較すると 8bit 分だけ精度が劣る．しかし，計算時間は数倍早く実用的な方法である [5]．**図 1** に4倍精度加算演算および乗算を double-double アルゴリズムを利用して計算するためのプログラムを示す．この図から加算および乗算の演算数はそれぞれ 11，24 であることが確認できる．そのため，doube-double アルゴリズムを用いて BLAS を4倍精度化する際に演算の ASUM（加算）や SCAL（乗算）のように1種類の四則演算しか用いないものであれば，理論的に計算量は加算で 11 倍，乗算で 24 倍の計算時間になることが予想される．

また，乗算のアルゴリズムにおいて 134217729.0($=2^{27}+1$) を利用する演算は上位 26bit のデータと下位 26bit のデータに分割するオペレーションである (`p1`, `p2` はそれぞれ `ah` の上位 26bit, 下位 26bit のデータを格納している)．

```
subroutine dd_add(ch,cl,ah,al,bh,bl)
implicit none
double precision ch, cl, ah, al, bh, bl
double precision p1, p2, s1
p1 = ah + al
p2 = p1 - ah
cl = (ah -(p1-p2)) + (bh - p2)
ch = p1
cl = cl +al + bl
s1 = ch + cl
cl =cl - (s1 -ch)
ch =s1
end
```

(a) 4倍精度の加算 ($C = A + B$)

```
subroutine dd_mul(ch,cl,ah,al,bh,bl)
implicit none
double precision ch, cl, ah, al, bh, bl
double precision u1, p1, p2, r1, r2, s1
ch = ah *bh
u1 = 134217729.0*ah
p1 = u1 - (u1 - ah)
p2 = ah - p1
u1 = 134217729.0*bh
r1 = u1 - (u1 - bh)
r2 = bh - r1
cl = ((p1*r1-ch) + p1*r2 + p2*r1) + p2*r2
cl = cl + (ah * bl +al * bh)
s1 = ch +cl
cl =cl - (s1 -ch)
ch =s1
end
```

(b) 4倍精度の乗算 ($C = A \times B$)

**図 1** Bailey の double-double アルゴリズムを用いた加算と乗算のプログラム

### 2.2 4倍精度線形代数ルーチン群 QPBLAS

#### 2.2.1 QPBLAS の概要

原子力機構では基本線形代数ルーチン群（BLAS）を Bailey の double-double アルゴリズムを利用して4倍精度化し，QPBLAS (Quadrature Precision BLAS) として公開している [6]．

これに含まれる 40 個のルーチンを**表 1** に示す．BLAS，LAPACK はルーチン名の接頭辞に's'(single:単精度) や'd'(double:倍精度) をつけてその精度を表している．そこで，本ルーチン群では表 1 にあるように接頭辞に double-double アルゴリズムを表す'dd' を付けて表している．

また，このルーチン群のインターフェイスは**図 2** にあるように，基本的に BLAS の倍精度ルーチンの本来の引数の形に合わせている．しかし，本ルーチンでは4倍精度変数を上位データと下位データの2つの倍精度変数で表現するため，本ルーチンでは BLAS での倍精度の引数1つ当たり2つの倍精度の引数を指定することになる．

**表 1** QPBLAS のルーチン一覧

| Level 1 | Level2 | Level3 |
|---|---|---|
| ddswap | ddgemv | ddgemm |
| ddscal | ddsymv | ddsymm |
| ddcopy | ddtrmv | ddsyr2k |
| ddaxpy | ddtrsv | ddsyrk |
| dddot | ddsyr | ddtrmm |
| ddnorm2 | ddsyr2 | ddtrsm |
| ddsum | ddgbmv | ddzhemm |
| ddidmax | ddger | ddzher2k |
| ddrot | ddsmbv | ddzherk |
| ddrotg | ddtbmv | |
| ddrotm | ddtbsv | |
| ddrotmg | ddzgerc | |
| ddzdotc | ddzgeru | |
| ddzdotu | ddzhbmv | |
| | ddzhemv | |
| | ddzher | |
| | ddzher2 | |

#### 2.2.2 QPBLAS の計算性能

ここでは QPBLAS の代表的なルーチンである dddot(内積計算)，ddgemv(行列ベクトル積)，および ddgemm(行列行列積) の性能を調査する．この性能調査では，Intel および AMD のプロセッサを利用した．プロセッサ用の詳細は**表 2** に記す．このアルゴリズムは，計算順序を変化させる最適化を行うと，適正な計算ができない．そのため，強

```
  call dgemm(TRANSA,TRANSB,M,N,K,
&            ALPHA,A,LDA,B,LDB,BETA,C,LDC)
```

(a) 倍精度 BLAS(DGEMM)

```
  call ddgemm(TRANSA,TRANSB,M,N,K,
&             ALPHAH,ALPHAL,AH,AL,LDA,BH,BL,
&             LDB,BETAH,BETAL,CH,CL,LDC)
```

(b) 4 倍精度 BLAS (DDGEMM)

**図 2** BLAS と QPBLAS の行列行列積ルーチン (dgemm および ddgemm) のインターフェイス部分．QPBLAS において'*H' および' *L' が表す変数は 4 倍精度変数の上位データ，下位データをそれぞれ表現している．

**表 2** 性能評価に使用した計算環境.

(a) Intel/ Windows

| Intel | AMD |
|---|---|
| Processor | Intel Core2 Duo E8400 (3.0GHz) |
| OS | Windows XP Professional |
| Compiler | Intel Fortran 10.0 IA32 |

(b) AMD / Linux

| | |
|---|---|
| Processor | Dual Core AMD Opteron Processor 2800 (2.4GHz) |
| OS | Cent OS 4.4 |
| Compiler | gfortran 4.1.0 |

い最適化を用いる場合には計算順序を変化させないコンパイルオプションを利用する必要があることに注意が必要である．

**図 3**(a) に DDDOT の計算時間を示す．この計算における最適化オプションとして，Intel(Windows) では (/O0，/O2，/O3) を，AMD(Linux) では (-O0，-O2，-O3) を使用して比較している．また，最適化の強さに合わせて，計算順序を変化させないコンパイルオプションも利用している (Intel 版の/O2，/O3 の時に/Op(Lunix 版の-mp に対応) を使用している)．この結果から DDDOT の計算時間は配列（ベクトルの次元の増加に伴って線形に増加していることおよび AMD を利用した方が若干高速であること確認できる．また，DDOT との計算時間の比較を図 3(b) に示す．この結果から，ある程度の最適化を利用することで Intel では約 14 倍，AMD では約 3 倍の計算時間の増加になっているが，今回結果は掲載していないが，DDOT の計算時間は Intel の方が AMD より約 4 倍程度早い．このことから，DDDOT のアルゴリズムは AMD のプロセッサに適していると考えられる．

次に，DDGEMV(TRANS='N') の計算結果を**図 4** に示す．この結果から，配列の次元に 2 乗で比例していること，および，最適化を行った場合，Intel の方が高速に計算できることが確認できる．また，DGEMV との比較において Intel（最適化：O3）で約 8 倍，AMD は約 6 倍ではある．これは DGEMV の AMD での計算時間が大きいためである．

最後に，DDGEMM(TRANSA='N'，TRANSB='N') の結果を**図 5** に示す．この結果から，配列の次元に 3 乗で比例していることが確認できる．こと，および，最適化を行った場合，Intel の方が高速に計算できることが確認できる．また，DGEMM との比較の結果から GEMV との比較とほぼ同様な結果（Intel（最適化：O3）で約 8 倍，AMD は約 6 倍）であった．

さらに，行列行列積 (GEMM) での計算速度 (FLOPS) を**図 6** に示す．この時の最適化はそれぞれ O3(Intel) および O2(AMD) である．また，4 倍精度ルーチンの計算速度は倍精度演算を基に測定している．この結果からどちらのプロセッサでも 4 倍精度化することで計算性能が向上していることがわかる．これは，4 倍精度化により演算の密度が増加しているためであると考えられる．また，Intel の方がより高速であることが確認できる．

## 3. 4倍精度固有値計算ルーチン QPEigenK

### 3.1 QPEigenK の概要

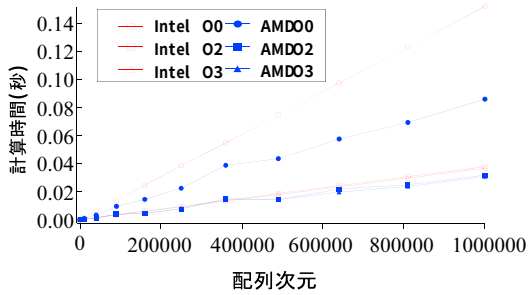
我々はこれまでに大規模並列計算機用の固有値ソルバを開発している [1], [7]．これらのコードを用いて実際に 375,000 次元の実対称行列の全固有値および固有ベクトルの計算に成功している．さらに，この成果を基に「京」コンピュータのアーキテクチャを考慮して開発した固有値ソルバを開発し，EigenK として公開している[*1][8]．この EigenK は実対称行列用およびエルミート行列用のアルゴリズムとして，通常の固有値計算で利用されている様にハウスホルダー変換で行列を 3 重対角行列に変換し，分割統治法で（3 重対角行列の）固有値・固有ベクトルを求め，逆変換で元の行列の固有ベクトルを求める方法を採用している．さらに，実対称行列用として 5 重対角行列を経由して固有値計算を行うコードも用意している．このコード群は既に「京」上でのシミュレーションに利用されている [9].

しかしながら，次節の大規模並列計算の結果から，行列の次元が 10 万次元程度でも得られる固有値の精度は 6 桁程度になることが示されており，「京」の性能を全て利用するような規模の計算では，得られる結果の精度に問題が出てくる可能性がある．そのため，高精度での計算が望まれていた．そこで，この EigenK の実対称行列用 3 重対角経由ソルバを基に QPBLAS を適用させ 4 倍精度化を行った [5]．この際，BLAS だけではなく，lapack, scalapack および MPI の利用しているルーチンも 4 倍精度化している．

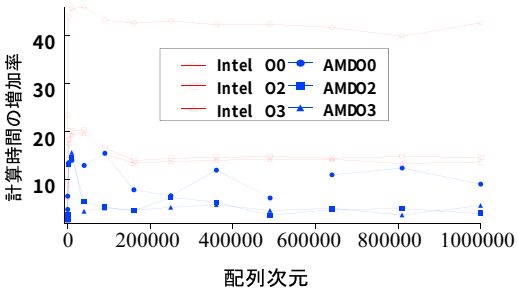
### 3.2 QPEigenK の計算性能調査

ここでは原子力機構の富士通 PRIMERGY BX900 を

[*1] JST からの CREST 受託研究「超伝導新奇応用のためのマルチスケール・マルチフィジックスシミュレーションの基盤構築」（研究期間：平成 18 年 10 月〜平成 24 年 10 月）によって開発されたものである．

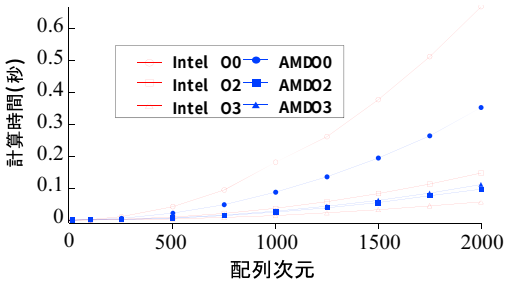



(a)dddot の計算時間

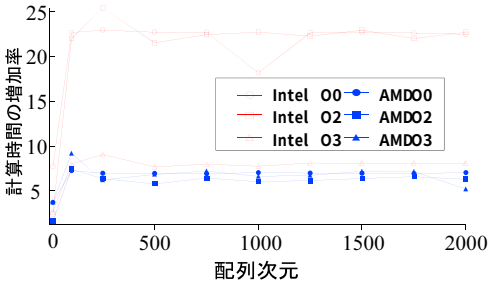



(b)dddot と dddot の計算時間の比較

**図 3** 4 倍精度内積計算（dddot）の計算時間.

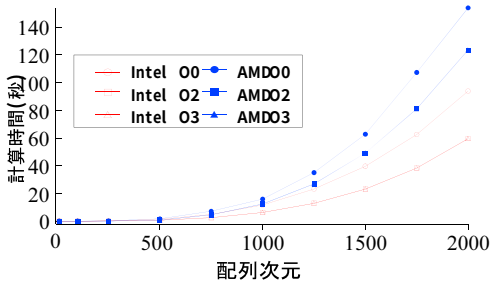



(a)ddgemv の計算時間

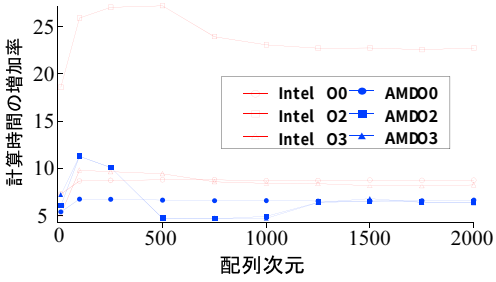



(b)ddgemv と dgemv の計算時間の比較

**図 4** 4 倍精度行列ベクトル積 (ddgemv）の計算時間.




(a)ddgemm の計算時間




(b)ddgemm と dgemm の計算時間の比較

**図 5** 4 倍精度行列行列積 (ddgemm）の計算時間.

利用して QPEigenK を計算した結果を紹介する．BX900 の性能等を**表 3** に示す．本計算では最適化オプションとして`-Kfast` を使用するが，演算順序を変化させないため `-Knoeval` も同時に適用する．今回の計算では固有値の理論解がわかる Frank 行列の全ての固有値・固有ベクトルを計算した．誤差の指標としては

$$Error = \max_i \left\{ abs\left( \frac{\bar{\lambda_i} - \lambda_i}{\bar{\lambda_i}} \right) \right\},$$

を採用する．ここで，$\bar{\lambda_i}$ および $\lambda_i$ はそれぞれ $i$ 番目に大きい理論固有値，数値計算により得られた固有値である．

**図 7** に倍精度固有値ソルバ EigenK および 4 倍精度固有値ソルバ QPEigenK を用いて Frank 行列を計算した際の行列サイズと固有値の精度の関係を示す．この結果から，行列サイズが 50,000 次元の場合，倍精度演算では 6 桁程度の精度しか得られないが，4 倍精度演算を行うことで 25 桁程度の精度が得られることが確認できる．このことから，計算機の大規模化に伴って行列が大規模化した場合，高精度の固有値計算のためには 4 倍精度演算が有力な 1 つ手法

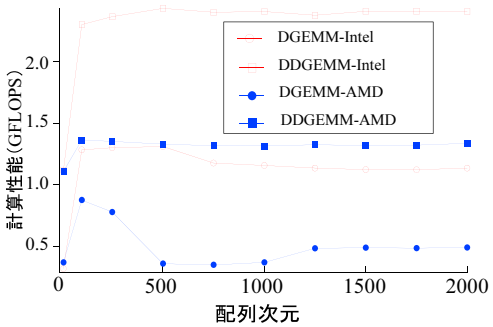



**図 6** 倍精度行列行列積 (dgemm) と 4 倍精度行列行列積 (ddgemm) の計算性能の比較.

であると結論付けられる．

次に，**図 8** に計算時間を**表 4** に計算性能を示す．この結果から，EigenK および QPEigenK のどちらも行列の次元が増加すると並列化の効果がより得られることが確認できるが，QPEigenK の方がより効果が得られていることが確認できる．これは 4 倍精度化による通信コストの増加率よりも，計算コストの増加率が多く，通信の影響が少なく

**表 3** 富士通 PRIMERGY BX900 の性能.

| | BX900 |
|---|---|
| Processor | Intel Xeon X5570<br>Quad core (2.93GHz) |
| Number of processors per each node | 2 (8 cores) |
| Number of total nodes | 2134 |
| Network | InfiniBand QDR |
| Bandwidth(inter-node) | 8.0GB/s full-duplex |
| Compiler<br>Compile option | Fujitsu Fortran Compiler<br>`-Kfast,noeval` |
| Library | SSL II V3.2<br>BLAS, LAPACK<br>ScaLAPACK |

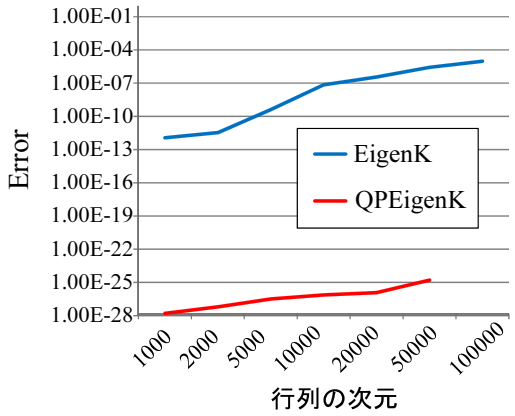
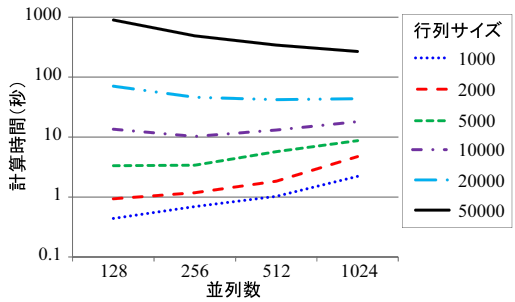
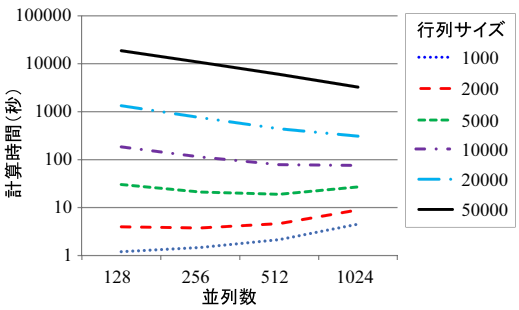



**図 7** EigenK および QPEigenK による Frank 行列の固有値の精度比較.




(a) EigenK の計算時間




(b) QPEigenK の計算時間

**図 8** EigenK および QPEigenK による Frank 行列の固有値計算の計算時間の比較.

**表 4** 固有値計算ルーチンの計算性能.

(a)EigenK の計算性能

| Dim. of Matrix | GFLOPS(ピーク性能比) | | | |
|---|---|---|---|---|
| | 128PE | 256PE | 512PE | 1024PE |
| 1000 | 22.15 ( 1.5 %) | 15.52 ( 0.5 %) | 14.59 ( 0.2 %) | 8.21 ( 0.1 %) |
| 2000 | 58.78 ( 3.9 %) | 48.96 ( 1.6 %) | 42.29 ( 0.7 %) | 18.25 ( 0.2 %) |
| 5000 | 197.66 (13.2 %) | 198.05 ( 6.6 %) | 140.43 ( 2.3 %) | 95.82 ( 0.8 %) |
| 10000 | 343.88 (22.9 %) | 456.97 (15.2 %) | 397.58 ( 6.6 %) | 294.34 ( 2.5 %) |
| 20000 | 489.97 (32.7 %) | 754.80 (25.2 %) | 887.58 (14.8 %) | 860.45 ( 7.2 %) |
| 50000 | 564.32 (37.6 %) | 1041.76 (34.7 %) | 1533.46 (25.6 %) | 1980.84 (16.5 %) |
| 100000 | — ( — %) | — ( — %) | 1739.16 (29.0 %) | 2882.22 (24.0 %) |

(b)QPEigenK の計算性能

| Dim. of Matrix | GFLOPS(ピーク性能比) | | | |
|---|---|---|---|---|
| | 128PE | 256PE | 512PE | 1024PE |
| 1000 | 123.00 ( 8.2 %) | 107.72 ( 3.6 %) | 100.86 ( 1.7 %) | 53.44 ( 0.4 %) |
| 2000 | 213.51 (14.2 %) | 235.85 ( 7.9 %) | 247.20 ( 4.1 %) | 137.52 ( 1.1 %) |
| 5000 | 332.11 (22.1 %) | 487.13 (16.2 %) | 628.77 (10.5 %) | 459.97 ( 3.8 %) |
| 10000 | 390.14 (26.0 %) | 638.78 (21.3 %) | 1009.21 (16.8 %) | 1069.57 ( 8.9 %) |
| 20000 | 407.00 (27.1 %) | 726.30 (24.2 %) | 1302.51 (21.7 %) | 1867.36 (15.6 %) |
| 50000 | — ( — %) | 765.74 (25.5 %) | 1386.80 (23.1 %) | 2582.56 (21.5 %) |

なったためである．また，50,000 次元の行列を 1024 個のコアで計算した場合，ピーク性能比は EigenK で 16.5%，QPEigenK で 21.5%であることからも，4 倍精度化により計算性能が向上していることが確認できる[*2]．そのため，この条件の場合，QPEigenK の計算時間は EigenK の約 12 倍であり，加算と乗算が 1 : 1 の割合で存在すると仮定したときの計算量の増加率である 17.5(=(11+24)/2) 倍よりも高速である．ただし，EigenK で 100,000 次元の行列を計算した際の性能は 24.0%であり，50,000 次元を計算した際の QPEigenK の性能を上回る．これは，今回のテストにおける行列を分割する際のブロックサイズが QPEigenK に適していなかったためであると考えているが，この点についての詳細は今後調査する予定である．

## 4. まとめ

本研究では原子力機構で開発した 4 倍精度 BLAS および

[*2] 前章と同様に 4 倍精度演算の計算速度は倍精度演算を基準に測定している．

これを利用して開発した 4 倍精度固有値ソルバ QPEigenK について紹介を行った．これらのルーチンの 4 倍精度化には Bailey の double-double アルゴリズムを利用して開発している．QPBLAS の代表的なルーチンの性能評価から，倍精度版と比較して 10 倍程度の計算時間の増加で 4 倍精度演算が可能であることを確認した．また，倍精度固有値ソルバでは 50,000 次元の Frank 行列の固有値の精度が 6 桁程度しかないのに対して，QPBLAS を実際に利用して開発した 4 倍精度固有値ソルバ QPEigenK を用いることで計算時間は約 10 倍以上増加したが 25 桁程度の精度で計算できることを確認した．この結果から，今後大規模化する計算機の性能をフルに利用するようなシミュレーションでは，誤差の累積のため計算結果の信頼性に問題が生じるかもしれないが，これらのルーチンを利用することで回避することが可能になると思われる．

4 倍精度基本線形計算ライブラリ QPBLAS は http://ccse.jaea.go.jp/ja/download/qpblas.html において 2 条項 BSD ライセンスで公開している [6]．また，QPEigenK は公開のための整備を行っているため現在のところ一般ユーザの利用はできないが，基になった「京」コンピュータ用固有値計算ライブラリ EigenK は http://ccse.jaea.go.jp/ja/download/eigenk.html において 2 条項 BSD ライセンスで公開している [8].

**謝辞** 本研究の一部は，CREST(JST) および科研費（基盤研究（c）一般 23500056）の成果によるものです．

## 参考文献

[1] S. Yamada, T. Imamura, T. Kano, and M. Machida, High-Performance Computing for Exact Numerical Approaches to Quantum Many-Body Problems on the Earth Simulator, *Proc. of SC06* (2006)

[2] 小武守恒，藤井昭宏，長谷川秀彦，西田晃：反復法ライブラリ向け 4 倍精度演算の実装と SSE2 を用いた高速化，情報処理学会論文誌 コンピューティングシステム，Vol. 1, No. 1, pp. 73-84 (2008)

[3] The MPACK; Multiple precision arithmetic BLAS (MBLAS) and LAPACK (MLAPACK) http://mplapack.sourceforge.net.

[4] High-Precision Software Directory. http://crd-legacy.lbl.gov/~dhbailey/mpdist/.

[5] T. Imamura, S. Yamada, M. Machida, Preliminary Report for a High Precision Distributed Memory Parallel Eigenvalue Solver, Poster Presentation in SC12 (2012). http://sc12.supercomputing.org/schedule/event_detail.php?evid=post236.

[6] QPBLAS 4 倍精度 Basic Linear Algebra Subroutines(online), 入手先 ⟨http://ccse.jaea.go.jp/ja/download/qpblas.html⟩ (2012.11.05).

[7] Imamura, T., Yamada, S., and Machida, M., “Development of a high performance eigensolver on the peta-scale next generation supercomputer system”, Progress in NUCLEAR SCIENCE and TECHNOLOGY, Vol. 2, pp.643-650 (2011). [CD-ROM]

[8] 「京」コンピュータ用固有値計算ライブラリ EigenK(online), 入手先 ⟨http://ccse.jaea.go.jp/ja/download/eigenk.html⟩ (2012.11.05).

[9] Y. Hasegawa, J. Iwata, M. Tsuji, D. Takahashi, A. Oshiyama, K. Minami, T. Boku, F. Shoji, A. Uno, M. Kurokawa, H. Inoue, I. Miyoshi, M. Yokokawa, First-principles calculations of electron states of a silicon nanowire with 100,000 atoms on the K computer, *Proc. of SC11* (2011)