

倍精度と4倍精度の混合型反復法の提案

小 武 守 恒^{†1,†2} 藤 井 昭 宏^{†3}
 長 谷 川 秀 彦^{†4} 西 田 晃^{†5,†1}

CG 法などのクリロフ部分空間法の収束性は丸め誤差に大きく影響される。収束の改善を図るには高精度演算が有効であるが計算時間が多くかかってしまう。われわれは、開発中の反復解法ライブラリ Lis に 4 倍精度演算を実装し Intel SSE2 命令を用いて高速化を行った。その結果、倍精度演算の計算時間の約 4.5 倍程度で抑えることができた。今回われわれは、さらなる計算時間短縮のため反復解法の反復中で 4 倍精度演算をなるべく使わないような解法、すなわち倍精度と 4 倍精度の混合型反復解法を提案する。数値実験から 4 倍精度演算の反復解法よりも計算時間を大幅に減少させることが可能となった。また、MPI 環境では倍精度の反復法よりも速いことを示した。

A proposal of mixture type iterative method of double precision and quadruple precision

HISASHI KOTAKEMORI, AKIHIRO FUJII, HIDEHIKO HASEGAWA
 and AKIRA NISHIDA

The convergence of Krylov subspace methods, including CG method etc., is much influenced by the rounding errors. The high precision operation is effective for the improvement of convergence, however the arithmetic operations are costly. We implemented the quadruple precision operations for iterative solver library Lis, and accelerated by using the Intel SSE2 instruction. The calculation time of our implemented quadruple operation is almost 4.5 times as long as of the current Lis double precision operations. In this paper, we propose the mixture type iterative method of double precision and quadruple precision that efficiently uses the quadruple precision operations in order to reduce the calculation time. The proposed method is shown to dramatically reduce calculation times than the iterative method of the quadruple precision operations by numerical experiments. We also showed that the proposed method might be faster than the double precision iterative method in the MPI environment.

1. はじめに

反復解法として CG, BiCG 法などのクリロフ部分空間法がよく使われている。CG 法は、理論的には高々 n 回 (n は係数行列の次元数) の反復で収束する。しかしながら、倍精度演算では丸め誤差の影響のため収束するまでに多くの反復回数が必要となったり、停滞したりする。収束の改善には高精度演算、例えば 4 倍

精度演算が有効であるが計算時間が多くかかってしまう¹⁾。

われわれは、開発中の反復解法ライブラリ Lis に倍精度浮動小数点数を 2 個用いた”double-double”精度を使用した 4 倍精度演算を実装した。倍精度とインタフェースを共通化するため係数行列 A と解 x と右辺 b は倍精度とし、ライブラリ内部で 4 倍精度を用いることにした。倍精度と 4 倍精度の混合演算に対して SSE2 命令を用いて高速化を行った結果、倍精度演算の計算時間の約 4.5 倍程度で抑えることができた²⁾。

今回、さらなる計算時間短縮のため反復解法の反復中で 4 倍精度演算をなるべく使わないような解法、すなわち倍精度と 4 倍精度の混合型反復解法を提案し、その有効性を確かめる。以下 2 節で反復解法ライブラリ Lis, 3 節で 4 倍精度演算の実装, 4 節で倍精度と 4 倍精度の混合型反復解法, 5 節で数値実験, 6 節で関連研究, 7 節でまとめを述べる。

†1 科学技術振興機構 戦略的創造研究推進事業
 JST CREST

†2 東京大学 情報理工学系研究科
 Grad. Sch. of Info. Sci. & Tech., the Univ. of Tokyo

†3 工学院大学
 Kogakuin University

†4 筑波大学 図書館情報メディア研究科
 Grad. Sch. of Lib., Info. & Media Stud., U. of Tsukuba

†5 中央大学 21 世紀 COE プログラム
 21st Century COE Program, the Chuo Univ.

2. 反復解法ライブラリ Lis

Lis³⁾ (a Library of Iterative Solvers for linear systems) は大規模疎行列係数の線型方程式 $Ax = b$ に対する反復法ライブラリで、C 言語と Fortran 90 で記述されている。逐次版、OpenMP 共有メモリ並列版、MPI 単独、あるいは OpenMP + MPI のハイブリッド分散メモリ並列版がある。

Lis には以下のような特徴がある：

- 12 通りの反復解法、8 通りの前処理が組み合わせて利用できる
- 11 通りの疎行列格納形式が利用できる
- 逐次と並列ともに共通のインタフェースで処理できる。逐次環境から並列環境への移行はプログラムの変更なし（あるいはごくわずかの変更）でよい
- 倍精度と 4 倍精度も共通のインタフェースで利用できる

現在、<http://ssi.is.s.u-tokyo.ac.jp/lis/> においてバージョン 1.1-beta1 を公開中である。

3. 4 倍精度演算の実装

3.1 4 倍精度基本演算

Bailey⁴⁾ は倍精度浮動小数点数（倍精度浮動小数と省略）を 2 個用いた “double-double” 精度のアルゴリズムを開発している。このアルゴリズムでは double-double 精度浮動小数 a を $a = a.hi + a.lo$, $|a.hi| > |a.lo|$ ($a.hi$ と $a.lo$ は倍精度浮動小数) として 4 倍精度を実現している。このアルゴリズムは Dekker⁵⁾ を基にしている。“double-double” 精度の利点として、倍精度浮動小数への変換は $a.hi$ を返すだけでよいことがあげられる。しかし、IEEE 準拠の 4 倍精度⁶⁾ の表現形式では仮数部が 112 ビットであるのに対して、倍精度浮動小数を 2 個利用しているため仮数部が 104 ビットと 8 ビット少なくなっている。

われわれは、このアルゴリズムを採用して 4 倍精度を実装した。Lis の倍精度とインタフェースを共通化するため、係数行列 A と解 x と右辺 b は倍精度浮動小数とし、ライブラリ内部で 4 倍精度を用いる。このため、ユーザが直接 4 倍精度の変数を扱うことはなく、IEEE 準拠の 4 倍精度と表現が多少違っていても影響は少ない。IEEE 倍精度演算は仮数部が 52 ビットであるのに対して、仮数部を 104 ビットとした高速な高精度演算によって反復解法の収束を改善するのが目的である。以下にこのアルゴリズムの概要を述べる。

全ての演算は IEEE 倍精度で round-to-even 丸めと

仮定する。 $a + b$ の倍精度浮動小数加算結果を $fl(a + b)$ と表す。 $err(a + b)$ を $a + b = fl(a + b) + err(a + b)$ とする。倍精度浮動小数乗算 $a \times b$ についても同様とする。

(1) FAST_TWO_SUM は $s = fl(a + b)$, $e = err(a + b)$ を計算する。ただし $|a| \geq |b|$ とする。

```
FAST_TWO_SUM(a,b,s,e) {
    s = a + b
    e = b - (s - a) }
```

(2) TWO_SUM は $s = fl(a + b)$, $e = err(a + b)$ を計算する。(1) と異なり $|a| \geq |b|$ を仮定していない。

```
TWO_SUM(a,b,s,e) {
    s = a + b
    v = s - a
    e = (a - (s - v)) + (b - v) }
```

(3) SPLIT は倍精度浮動小数 a を $a = h + l$ に分割する。ただし、 h は a の仮数部の最初の 26 ビット分を持ち l は残りの 26 ビット分を持つ。

```
SPLIT(a,h,l) {
    t = 134217729.0 * a
    h = t - (t - a)
    l = a - h }
```

(4) TWO_PROD は $p = fl(a \times b)$, $e = err(a \times b)$ を計算する。

```
TWO_PROD(a,b,p,e) {
    p = a * b
    SPLIT(a,ah,al)
    SPLIT(b,bh,bl)
    e = ((ah*bh-p)+ah*bl+al*bh)+al*bl }
```

(1) から (4) を用いることで以下の 4 倍精度加算と乗算が可能となる。

(5) ADD は 4 倍精度加算 $a = b + c$ を計算する。ただし、 $a = (a.hi, a.lo)$, $b = (b.hi, b.lo)$, $c = (c.hi, c.lo)$ である。

```
ADD(a,b,c) {
    TWO_SUM(b.hi,c.hi,sh,eh)
    TWO_SUM(b.lo,c.lo,sl,el)
    eh = eh + sl
    FAST_TWO_SUM(sh,eh,sh,eh)
    eh = eh + el
    FAST_TWO_SUM(sh,eh,a.hi,a.lo) }
```

(6) MUL は 4 倍精度乗算 $a = b \times c$ を計算する。

```
MUL(a,b,c) {
    TWO_PROD(b.hi,c.hi,p1,p2)
    p2 = p2 + (b.hi * c.lo)
    p2 = p2 + (b.lo * c.hi)
    FAST_TWO_SUM(p1,p2,a.hi,a.lo) }
```

3.2 4 倍精度ベクトルの格納方法

4 倍精度ベクトルのデータ構造としては次の 2 通りが考えられる (hi は double-double 精度浮動小数 $a = a.hi + a.lo$ の $a.hi$, lo は $a.lo$)。

- hi と lo を交互に格納する

- hi と lo を別の配列に格納する

今回われわれは、4 倍精度のベクトルを倍精度のベクトルとして利用することも考えている。後者の方法では、hi を格納している配列のみ用いれば倍精度浮動小数として扱うことができる。前者の方法では、倍精度のベクトルとして利用する場合、hi の値が 1 つおきとなるため行列ベクトル積 (matvec)、ベクトルの内積 (dot)、ベクトルおよびその実数倍の加減 (axpy) 等のプログラムの修正が必要になる。したがって、今回は matvec, dot, axpy 等がそのまま利用できる後者の方法を採用した。

3.3 Lis への実装

クリロフ部分空間法システムの反復解法は matvec, dot, axpy で実現されている。これらを 4 倍精度演算に置き換える。ただし、これまでと同じインタフェースで 4 倍精度演算が利用できるよう

- 係数行列 A , 右辺ベクトル b は倍精度
- 解ベクトル x の入出力は倍精度、反復解法中では 4 倍精度
- 反復解法中のベクトルとスカラーは 4 倍精度とした。これにより、matvec では倍精度 $\odot$ 4 倍精度、dot, axpy, スカラー・スカラー演算では 4 倍精度 $\odot$ 4 倍精度が必要となる ($\odot$ は対応する演算子)。matvec, dot, axpy の主な処理は積和演算である。4 倍精度の積和演算関数 FMA と混合精度 (4 倍精度と倍精度) の積和演算関数 FMAD を作成した。FMA は dot と axpy で、FMAD は matvec で利用される。

3.4 SSE2 による実装

SSE2 は Intel の Pentium4 に搭載された x87 命令に代わる高速化命令であり、128bit のデータに対して SIMD 処理を行える (64bit 倍精度浮動小数なら同時に 2 つの倍精度演算を行える)。SSE2 の利用には SSE2 の組み込み関数を利用する。

4 倍精度演算関数 ADD, MUL, FMA 等に対して SSE2 の組み込み関数を用いただけでは、計算の依存関係のため全体の約 50%程度しか SSE2 の packed-double 命令で処理できず、高速化の効果が不十分である。

実際に FMA を使用するのは dot, axpy, matvec であり、そこではループ内に 1 回ずつ FMA が使われていることに注目した。2 段のループアンローリングを行うとループ内で FMA が 2 回となり、すべて SSE2 の packed-double 命令で同時に処理できる。2 個同時に積和演算をする関数 FMA2_SSE2, FMAD2_SSE2 を用意して、それらを使う関数 axpy_fma2, dot_fma2, matvec_fma2 を作成した。

3.5 並列化

分散メモリでの MPI 並列を考える。行列とベクトルを行ブロック分割し、各プロセッサ (PE) は連続した行ブロックを持つこととする。通信が発生するのは、matvec と dot の処理時である。

matvec では、matvec の開始前に自分の PE に含まれないベクトルの要素を他 PE から取得する。各 PE のベクトル要素の通信には MPI_Isend と MPI_Irecv を用いた 1 対 1 通信を行う。この通信を効率よく行うために行列作成時に通信テーブルを作成する。送信は送信バッファにすべてを格納してから各 PE に送信している。また、4 倍精度ベクトルの hi と lo は別々に送信するのでなく、送信バッファに hi と lo の両方を格納して 1 度に送信している。

dot では、dot の最後に MPI_Allreduce を用いて各 PE で計算した値の総和を求める。4 倍精度の場合、MPI 標準の演算子は利用できない。そこで、4 倍精度用のデータ構造と演算子を用意した。図 1 に 4 倍精度の総和ルーチンのコードを示す。図 1 の lis_initialize の中で MPI_DOUBLE 2 個分の長さを持つ型 LIS_MPI_QDOUBLE と 4 倍精度の総和を行う関数 lis_mpi_qsum の登録を行っている。

```
typedef struct {
    double hi;
    double lo;
} LIS_QDOUBLE;

int lis_initialize(int argc, char* argv[]) {
    ...
    MPI_Type_contiguous( 2, MPI_DOUBLE,
        &LIS_MPI_QDOUBLE );
    MPI_Type_commit( &LIS_MPI_QDOUBLE );
    MPI_Op_create((MPI_User_function *)
        lis_mpi_qsum, LIS_TRUE, &LIS_MPI_QSUM);
    ...
}

void lis_mpi_qsum(LIS_QDOUBLE *invec,
    LIS_QDOUBLE *inoutvec, int *len,
    MPI_Datatype *datatype) {

    for(i=0;i<*len;i++)
        ADD(inoutvec[i],inoutvec[i],invec[i]);
}
```

図 1 4 倍精度の総和のコード。

Fig. 1 Code of summation for quadruple precision.

4. 倍精度と 4 倍精度の混合型反復解法

クリロフ部分空間法では丸め誤差の影響により、倍精度演算では収束までに多くの反復が必要となったり、停滞したりする。収束の改善には高精度演算が有効で

あるが、これまでは倍精度演算に比べて 20 倍もの計算時間が必要なため非現実的だった²⁾。今回、係数行列 A と解 x と右辺 b は倍精度、その他は 4 倍精度として SSE2 命令を用いた高速な実装を行ったが、それでも倍精度の約 4.5 倍の計算時間が必要である。さらに計算時間を短縮するためには、反復すべてを 4 倍精度演算で行うのではなく、必要なときだけ 4 倍精度演算を利用することが重要である。今回われわれは、倍精度と 4 倍精度の混合型反復法として 2 つの手法を提案する。

- SWITCH
- PERIODIC

SWITCH は、倍精度で解いた解（あるいは途中の値）を初期値として 4 倍精度で解く（演算精度を変えたりスタート）手法である。リスタート時に渡すのは倍精度演算で生成された解 x_k のみで、その他の中間変数は渡さない。このアルゴリズムを図 2 に示す。nrm2 は相対残差ノルムで、restart_tol はリスタートするための判定基準である。

```
for(k=0;k<最大反復回数;k++) {
    倍精度演算の反復解法
    if( nrm2<restart_tol ) break;
}
x 以外の作業変数をゼロクリア
for(k=k+1;k<最大反復回数;k++) {
    4 倍精度演算の反復解法
}
```

図 2 SWITCH のアルゴリズム。
Fig. 2 Algorithm for SWITCH.

PERIODIC は、倍精度 10 回に対して 4 倍精度 1 回のように一定間隔ごと 4 倍精度で反復を行う手法である。この手法は、4 倍精度演算の使用回数を容易にコントロールできる。精度切り替え時にすべての変数を渡す。ベクトル、スカラーは 4 倍精度で作成するため、4 倍精度から倍精度への切り替えはコストなしで行える。逆に、倍精度から 4 倍精度への切り替えは反復計算に必要なものだけ 1o をゼロクリアする。このアルゴリズムを図 3 に示す。

5. 数 値 実 験

今回実装した倍精度と 4 倍精度の混合型反復解法の性能を評価するために

- Lis の倍精度 (DOUBLE)
- Lis の 4 倍精度 (QUAD)
- 混合精度 (SWITCH)
- 混合精度 (PERIODIC)

```
for(k=0;k<最大反復回数;k++) {
    if( k%interval<num ) {
        4 倍精度演算の反復解法
    } else {
        1o をゼロクリア
        倍精度演算の反復解法
    }
}
```

図 3 PERIODIC のアルゴリズム。
Fig. 3 Algorithm for PERIODIC.

表 1 計算環境
Table 1 Evaluation platforms.

Machine	PC Cluster
CPU	Xeon
Clock	2.8GHz
L1D Cache	8KB
L2 Cache	512KB
Memory	1GB
Network	Gigabit Ethernet
OS	Linux
Compiler	Intel C/C++7.0
Options	-O3

を前処理なしの BiCG 法と BiCGSTAB 法を用いて比較した（4 倍精度に対応した前処理がまだ実装できていないため、前処理は用いていない）。

係数行列 A は Toeplitz 行列（次数： 10^5 ）

$$A = \begin{pmatrix} 2 & 1 & & & & & \\ 0 & 2 & 1 & & & & \\ \gamma & 0 & 2 & 1 & & & \\ & \ddots & \ddots & \ddots & \ddots & & \\ & & \gamma & 0 & 2 & 1 & \\ & & & \gamma & 0 & 2 & \end{pmatrix} \quad (1)$$

と University of Florida Sparse Matrix Collection⁷⁾ から airfoil_2d（次数：14,214，非零要素数：259,688），wang3（次数：26,064，非零要素数：177,168），language（次数：399,130，非零要素数：1,216,334）を用いた。行列の格納形式は CRS⁸⁾を用いた。右辺ベクトル b はすべてを 1，初期ベクトル x_0 はすべてを 0 とした。収束判定基準は相対残差ノルムを 10^{-12} とした。数値実験は表 1 に示す環境で行った。

5.1 収束の改善

Toeplitz 行列に対する 1PE での実行結果を示す（ $\gamma = 1.3$ を表 2 に $\gamma = 1.4$ を表 3）。表中の ϵ は SWITCH のリスタート基準を、num は PERIODIC の num を、“total” は倍精度と 4 倍精度の反復回数の合計を、“(double)” は倍精度で反復した反復回数を、 $\|b - Ax\|$ は収束後に近似解 x_k を用いて改めて計算したノルムを、“_” は 1,000 回の反復で収束しなかった

ことを意味する．また，PERIODIC の interval は 10 とした．したがって，10 反復の内 4 倍精度反復が num 回となる．実行時間の単位は秒で，各反復解法で一番速く収束した結果を太字で示している．この問題は倍精度演算では収束しなかった．

これらの表から以下のことが分かる：

- SWITCH は QUAD と比較して，BiCG で最大 3.95 倍の高速化，BiCGSTAB で最大 4.50 倍の高速化
- SWITCH は ϵ の値を小さくするほど実効時間が短縮される傾向にあるが， $\gamma = 1.4$ に対して BiCG は 10^{-8} より小さくなると収束しない
- PERIODIC は QUAD と比較して BiCG で最大 1.66 倍の高速化，BiCGSTAB で最大 1.90 倍の高速化
- PERIODIC は γ の値が大きくなると収束しないことが多く，num の値は大きくなる必要がある

倍精度 BiCG 法の収束履歴を図 4 に示す．この図から， $\gamma = 1.3$ の場合は残差ノルムが 10^{-11} あたりまで減少した後，停滞している．同様に， $\gamma = 1.4$ の場合は残差ノルムが 10^{-9} あたりまで減少した後，停滞している．残差ノルムが停滞してから 4 倍精度に切り替えても効果がないことが分かる．したがって，SWITCH の活用には停滞をうまく検出してリスタートを行うことが好ましい（実際は難しい）．

PERIODIC は interval=10 で固定しているので，num の値が大きくなるほど 4 倍精度で反復する割合が増加する．途中で 4 倍精度演算を導入するだけでは，倍精度演算に戻した時点で有用な情報が失われ，収束の改善にならないことが分かる．

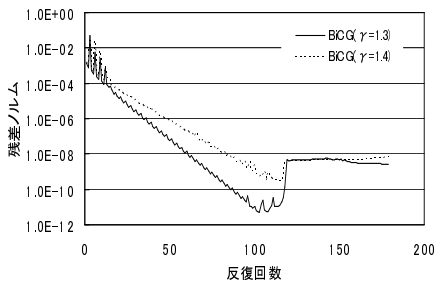


図 4 Toeplitz 行列に対する倍精度 BiCG 法の収束履歴.
Fig. 4 Convergence history of double precision BiCG method for Toeplitz matrix.

次に，行列 airfoil.2d，行列 wang3，行列 language に対する BiCG 法と BiCGSTAB 法の 1PE での実行結果を表 4 から表 6 に示す．表中の inter は PERI-

ODIC の interval を意味する．また，PERIODIC の num は 1 とした．したがって，inter 反復の内 4 倍精度反復が 1 回となる．

これらの表から以下のことが分かる：

- 実行時間は DOUBLE が最も少ないが，解の精度は QUAD と SWITCH は DOUBLE より最大 2 桁向上している
- SWITCH は QUAD と比較して BiCG で最大 3.20 倍の高速化，BiCGSTAB で最大 4.63 倍の高速化

DOUBLE の収束判定基準を変化させた場合の結果を表 7 に示す．この表より倍精度では収束判定基準を小さくしても反復回数が増加するだけで収束した解の品質 $\|b - Ax\|$ の値はほとんど向上しない．一方，最適な SWITCH を用いれば倍精度の実行時間の 2 割増程度で 2 桁も精度のよい解が得られる．収束判定基準を 10^{-12} ， $\epsilon = 10^{-12}$ として倍精度計算が収束した後に 4 倍精度による反復を行うことにより比較的少ないコストで解の改善が可能である．

以上より，SWITCH は倍精度で収束しない問題と収束する問題ともに QUAD より高速で有効な手法である．特に，倍精度で収束する問題に対しては QUAD と同様，DOUBLE より最大 2 桁も解の精度が向上している．問題は，倍精度で収束しない問題に対してリスタート基準の決定を誤ると収束しなくなったり収束までに時間がかかってしまうことである．PERIODIC は 4 倍精度の回数を容易にコントロールできるため，倍精度で収束する問題に対しては SWITCH より高速であるが，解の精度については DOUBLE と同程度である．また，倍精度が収束しない問題だと 4 倍精度の反復の割合を小さくすると数値的に不安定になる．

5.2 並列化

DOUBLE と QUAD の並列性能を調べるために，Toeplitz 行列を BiCG 法で 100 反復したときの 1PE から 8PE までの結果を表 8 に示す．

この表から以下のことが分かる：

- DOUBLE より QUAD の speed-up がよい
- DOUBLE と QUAD の ratio は PE が増加するごとに減少している

これは，4 倍精度演算はデータの通信よりも演算の割合が大きいため，PE を多くしたときに通信の影響を受ける割合が少なくなっているためだと思われる．そのため，QUAD と DOUBLE の ratio が縮まっている．

次に，Toeplitz 行列に対して $\gamma = 1.3$ としたときの BiCG と BiCGSTAB 法の 1PE から 8PE までの実行

結果を表 9 に示す。ただし、SWITCH と PERIODIC は 8PE で最適であった結果を掲載している。

この表から、PERIODIC は 1PE では収束しなくても PE 数を増やすと収束する場合があることが分かる。これは、ベクトルの分割によって dot の計算順序が変更され、丸め誤差の影響が収束する程度に軽減されたためだと思われる。

最後に、行列 `airfoil.2d` に対する実行結果を表 10 に示す。表中の “break” は反復中にブレイクダウンしたことを意味する。

この表から以下のことが分かる：

- PERIODIC は DOUBLE よりも少ない時間で収束することがある
- DOUBLE と SWITCH は PE 数を増やすとブレイクダウンする場合がある

PERIODIC は、反復回数が DOUBLE よりも少なく、PE 数が増加すると QUAD と DOUBLE の実行時間比が縮まるため、8PE では DOUBLE よりも少ない時間で収束したと思われる。また、ブレイクダウンしたのは、ベクトルの分割によって dot の計算順序が変更されたためだと思われる。

6. 関連研究

double-double 精度浮動小数演算ライブラリとして、QD ライブラリ⁹⁾がある。QD は C++ で記述されたライブラリである。さらに、倍精度浮動小数を 4 個用いた “quad-double” 精度のアルゴリズムを用いることで 8 倍精度演算も実現している。しかし、SSE2 等の SIMD 命令を用いた高速化は行われていない。

4 倍精度演算をサポートした反復解法ライブラリとしては、GMM++ (Generic Matrix Methods)¹⁰⁾がある。GMM++ は C++ で記述された密と疎行列に対するテンプレートライブラリで、4 倍精度演算については QD ライブラリを外部ライブラリとして呼び出して利用している。

Langou¹¹⁾らは倍精度演算の直接解法で得られた解を 4 倍精度演算で反復改良することで、4 倍精度演算の直接解法より高速に解を求めている。4 倍精度演算には FORTRAN の 4 倍精度を用いている。

本研究では直接解法ではなく反復解法を用いた倍精度と 4 倍精度の混合精度を提案している。また、Lis 倍精度浮動小数版と同一のインタフェースでライブラリの内部で double-double 精度を利用して SSE2 命令を用いて高速化している。Lis に特化した方法を用いているため、IEEE 標準とは異なるが現実的かつ高速な実装となっている。

7. ま と め

本稿では、“double-double” 精度を用いて反復解法ライブラリ Lis に 4 倍精度演算を実装した。倍精度とインタフェースを共通化するため四則演算に対して倍精度と 4 倍精度の混合演算を行い、SSE2 命令を用いて高速化を行った。さらに、収束と計算時間の短縮を行う倍精度と 4 倍精度の混合型反復法を提案した。

数値実験で、倍精度で解いた解（あるいは途中の値）を初期値として 4 倍精度で解く SWITCH と一定間隔ごと 4 倍精度で反復を行う PERIODIC とともに 4 倍精度演算の反復解法よりも計算時間を大幅に短縮できることを示した。QUAD と比較して SWITCH は最大 4.50 倍の高速化を、PERIODIC は最大 4.26 倍の高速化を実現した。さらに、SWITCH は倍精度で収束する問題に対しては倍精度の実行時間の 2 割増程度で最大 2 桁精度のよい解が得られることを示した。

MPI を用いた分散並列環境では、並列化によって倍精度と 4 倍精度の性能差が縮まる。このため、PERIODIC は 8PE では倍精度演算の反復法よりも実行時間が少なくなることがある。このことより、収束の困難な大規模問題で、PE 数が増えると倍精度と 4 倍精度の実行時間が逆転する可能性は少なくないだろう。

これらのことより、Lis に対して倍精度と高速に実装された 4 倍精度を混合して使うことにより最小限の追加計算コストで反復解法の収束の改善を図ることが可能となった。

今後の課題として、4 倍精度をうまく活用して収束性を向上、安定して解けるロバストなライブラリにするため、たとえば SWITCH の最適ナリスタート基準の決定法の開発などがあげられる。

参 考 文 献

- 1) H. Hasegawa. Utilizing the Quadruple-Precision Floating-Point Arithmetic Operation for the Krylov Subspace Methods. the 8th SIAM Conference on Applied Linear Algebra, 2003.
- 2) 小武守恒, 藤井昭宏, 長谷川秀彦, 西田晃. SSE2 を用いた反復解法ライブラリ Lis 4 倍精度版の高速化. 情報処理学会研究報告, 2006-HPC-108, pp. 7-12, 2006.
- 3) <http://ssi.is.s.u-tokyo.ac.jp/lis/>
- 4) D. H. Bailey. A fortran-90 double-double library. <http://www.nersc.gov/~dhbailey/mpdist/mpdist.html>.
- 5) T. Dekker. A floating-point technique for extending the available precision. Numerische

表 2 Toeplitz 行列 ($N = 10^5$, $\gamma = 1.3$) に対する BiCG 法と BiCGSTAB 法の収束性.

Table 2 Convergence of BiCG and BiCGSTAB methods for Toeplitz matrix.

		BiCG				BiCGSTAB			
		iter.		sec.	$\ b - Ax\ $	iter.		sec.	$\ b - Ax\ $
		total	(double)			total	(double)		
QUAD		113		6.60	2.47E-10	95		6.61	1.80E-10
SWITCH	$\epsilon = 10^{-9}$	95	(74)	2.33	2.53E-10	73	(63)	1.71	1.71E-10
	$\epsilon = 10^{-10}$	95	(86)	1.82	2.51E-10	82	(76)	1.63	1.06E-10
	$\epsilon = 10^{-11}$	103	(100)	1.67	9.34E-11	102	(100)	1.78	3.01E-10
PERIODIC	num = 1	—				172	(154)	3.90	2.71E-10
	num = 2	—				126	(100)	3.47	2.96E-10
	num = 3	—				114	(78)	3.82	2.35E-10
	num = 4	—				113	(66)	4.35	2.54E-10
	num = 5	107	(52)	3.98	3.16E-10	105	(50)	4.66	1.72E-10

表 3 Toeplitz 行列 ($N = 10^5$, $\gamma = 1.4$) に対する BiCG 法と BiCGSTAB 法の収束性.

Table 3 Convergence of BiCG and BiCGSTAB methods for Toeplitz matrix.

		BiCG				BiCGSTAB			
		iter.		sec.	$\ b - Ax\ $	iter.		sec.	$\ b - Ax\ $
		total	(double)			total	(double)		
QUAD		155		8.77	2.85E-10	135		9.40	2.69E-10
SWITCH	$\epsilon = 10^{-8}$	115	(83)	3.06	3.04E-10	89	(66)	2.67	2.45E-10
	$\epsilon = 10^{-9}$	—				103	(93)	2.20	1.23E-10
	$\epsilon = 10^{-10}$	—				120	(115)	2.18	3.00E-10
	$\epsilon = 10^{-11}$	—				124	(122)	2.09	1.92E-10
PERIODIC	num = 3	—				—			
	num = 4	—				—			
	num = 5	—				175	(85)	7.79	2.09E-10

表 4 行列 airfoil.2d に対する BiCG 法と BiCGSTAB 法の収束性.

Table 4 Convergence of BiCG and BiCGSTAB methods for airfoil.2d matrix.

		BiCG				BiCGSTAB			
		iter.		sec.	$\ b - Ax\ $	iter.		sec.	$\ b - Ax\ $
		total	(double)			total	(double)		
DOUBLE		4567	(4567)	18.64	3.25E-08	3410	(3410)	12.82	3.51E-08
QUAD		3838		69.39	5.36E-10	2835		58.07	4.42E-10
SWITCH	$\epsilon = 10^{-10}$	4402	(4091)	24.25	3.15E-10	3488	(3060)	21.58	4.28E-10
	$\epsilon = 10^{-11}$	4331	(4176)	21.66	3.13E-10	3405	(3150)	18.46	4.42E-10
	$\epsilon = 10^{-12}$	4709	(4567)	22.87	3.56E-10	3504	(3410)	15.98	3.96E-10
PERIODIC	inter = 40	4567	(4452)	21.81	3.11E-08	3324	(3240)	15.21	3.44E-08
	inter = 50	4093	(4011)	21.91	3.10E-08	3036	(2975)	13.63	3.37E-08
	inter = 60	4260	(4189)	22.17	3.05E-08	3772	(3709)	16.64	3.76E-08

表 5 行列 wang3 に対する BiCG 法と BiCGSTAB 法の収束性.

Table 5 Convergence of BiCG and BiCGSTAB methods for wang3 matrix.

		BiCG				BiCGSTAB			
		iter.		sec.	$\ b - Ax\ $	iter.		sec.	$\ b - Ax\ $
		total	(double)			total	(double)		
DOUBLE		476	(476)	2.03	3.52E-10	234	(234)	0.98	2.26E-10
QUAD		372		7.31	1.49E-10	228		5.25	1.44E-10
SWITCH	$\epsilon = 10^{-11}$	459	(444)	2.42	9.22E-11	229	(226)	1.15	1.45E-10
	$\epsilon = 10^{-12}$	479	(476)	2.32	1.46E-10	235	(234)	1.13	1.38E-10

Mathematik, vol.18 pp.224–242, 1971.

- 6) Intel Fortran Compiler User's Guide Vol I.
- 7) T. Davis. UF Sparse Matrix Collection. <http://www.cise.ufl.edu/research/sparse/matrices>.
- 8) R. Barrett, et al.. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, 1994.
- 9) Y. Hida, X. S. Li and D. H. Bailey. Algorithms for quad-double precision floating point arith-

metic. Proceedings of the 15th Symposium on Computer Arithmetic, pp.155–162, 2001.

- 10) Y. Renard and J. Pommier. GMM++ User Guide. http://www-gmm.insa-toulouse.fr/get-fem/gmm_intro.
- 11) J. Langou, et al.. Exploiting the Performance of 32 bit Floating Point Arithmetic in Obtaining 64 bit Accuracy (Revisiting Iterative Refinement for Linear Systems). SC06 Technical Paper, 2006.

表 6 行列 language に対する BiCG 法と BiCGSTAB 法の収束性.

Table 6 Convergence of BiCG and BiCGSTAB methods for language matrix.

	BiCG				BiCGSTAB			
	iter.		sec.	$\ b - Ax\ $	iter.		sec.	$\ b - Ax\ $
	total	(double)			total	(double)		
DOUBLE	39	(39)	3.42	2.96E-09	32	(32)	2.86	2.54E-09
QUAD	36		10.53	4.25E-11	29		9.52	1.99E-10
SWITCH $\epsilon = 10^{-11}$	37	(35)	4.07	4.20E-10	34	(31)	4.11	3.81E-10
$\epsilon = 10^{-12}$	40	(39)	4.18	4.47E-10	34	(32)	3.89	3.34E-10

表 7 行列 airfoil.2d に対する倍精度 BiCG 法と BiCGSTAB 法の収束.

Table 7 Convergence of double precision BiCG and BiCGSTAB methods for airfoil.2d matrix.

tol	BiCG			BiCGSTAB		
	iterations	sec.	$\ b - Ax\ $	iterations	sec.	$\ b - Ax\ $
10^{-12}	4567	18.64	3.25E-08	3410	12.82	3.51E-08
10^{-14}	4897	20.02	3.28E-08	3779	14.30	3.58E-08

表 8 DOUBLE と QUAD の並列性能 (Toeplitz 行列, BiCG 法を 100 反復).

Table 8 Parallel performance of DOUBLE and QUAD (Toeplitz matrix, 100 iterations for BiCG method).

PE	1	2		4		8	
	sec.	sec.	speed-up	sec.	speed-up	sec.	speed-up
DOUBLE	1.31	0.74	1.78	0.46	2.84	0.30	4.41
QUAD	5.77	2.92	1.97	1.50	3.83	0.91	6.36
ratio (QUAD/DOUBLE)	4.39	3.95		3.25		3.04	

表 9 Toeplitz 行列 ($N = 10^5$, $\gamma = 1.3$) に対する BiCG 法と BiCGSTAB 法の並列性能 (上段が BiCG, 下段が BiCGSTAB).

Table 9 Parallel performance of BiCG(up) and BiCGSTAB(bottom) methods for Toeplitz matrix.

PE		1		2		4		8	
		iter.	sec.	iter.	sec.	iter.	sec.	iter.	sec.
QUAD		113	6.60	113	3.30	113	1.70	113	1.00
SWITCH $\epsilon = 10^{-11}$		103	1.67	103	0.97	103	0.48	103	0.33
PERIODIC num = 4		—		113	1.95	114	1.04	113	0.63
QUAD		95	6.61	98	3.50	96	1.76	95	1.05
SWITCH $\epsilon = 10^{-10}$		82	1.63	91	1.00	84	0.50	82	0.39
PERIODIC num = 3		114	3.82	121	2.13	103	0.94	99	0.62

表 10 行列 airfoil.2d に対する BiCG 法と BiCGSTAB 法の並列性能 (上段が BiCG, 下段が BiCGSTAB).

Table 10 Parallel performance of BiCG(up) and BiCGSTAB(bottom) methods for airfoil.2d matrix.

PE		1		2		4		8	
		iter.	sec.	iter.	sec.	iter.	sec.	iter.	sec.
DOUBLE		4567	18.64	4398	10.81	4572	7.07	4399	5.70
QUAD		3838	69.39	3839	36.30	3839	19.62	3837	12.08
SWITCH $\epsilon = 10^{-12}$		4709	22.87	5029	17.47	4985	9.44	5039	7.72
PERIODIC inter = 60		4260	22.17	4090	11.09	4361	7.20	4092	5.45
DOUBLE		3410	12.82	break		3612	6.60	3330	6.55
QUAD		2835	58.07	2643	29.20	3016	18.62	2998	12.78
SWITCH $\epsilon = 10^{-10}$		3488	21.58	break		3928	9.98	3222	6.74
PERIODIC inter = 40		3324	15.21	3054	8.87	3607	7.20	3025	6.18