Vol.2018-HPC-167 No.31
2018/12/18

分散並列計算環境における行列・ベクトル積の高精度な実装と丸め誤差解析

小林 亮太[1,a)]　尾崎 克久[1]

概要：分散並列計算環境において行列・ベクトル積の高精度計算アルゴリズムを扱う．OpenMP と MPI の両方を使用した並列化に対応する高精度計算アルゴリズムを開発し，その丸め誤差評価を行う．分散並列計算環境において高速化するために，任意の順序で計算可能な高精度計算アルゴリズムや誤差上限付きアルゴリズムを提案する．また，PBLAS の行列・ベクトル積ルーチン PDGEMV に対応するように議論を拡張し，本報告の最後には京コンピュータと富士通 FX100 におけるパフォーマンスを紹介する．

Accurate Matrix-Vector Multiplication and Rounding Error Analysis for Parallel and Distributed Computing

Ryota Kobayashi[1,a)]　Katsuhisa Ozaki[1]

1. はじめに

本研究では，分散並列計算環境上で行列・ベクトル積を計算する高精度計算アルゴリズムを提案する．浮動小数点演算は丸め誤差の問題を抱えており，計算結果の精度を改善するために適宜高精度計算を適用することが有用である．本稿では，浮動小数点数を入力とし，アルゴリズム内部で高精度計算を行い，最後に浮動小数点数を返すアルゴリズムを扱う．逐次実行モデルのアルゴリズムとして，文献 [1] においてベクトルの内積に関する高精度計算アルゴリズムである Dot2 と DotK が提案されている．分散並列計算環境用アルゴリズムに関しては，文献 [2] にて Dot2 と DotK を拡張した PDotK というアルゴリズムが提案されている．

分散並列計算環境では，ノード内で OpenMP を，全体で MPI を用いることで高効率を達成できる．また，使用ノード数が多いときにはリダクション処理の改善も重要である．これらを達成するために，高精度計算アルゴリズムを逐次実行モデルから変更することが必要となる．よって本研究では，任意の計算順序に対応するよう，先行研究の Dot2 と DotK を一般化し，さらに，PBLAS の行列・ベクトル積ルーチン PDGEMV の形式に対応したアルゴリズムとその誤差解析を行った．最後に分散並列計算環境において実装を行った数値実験結果について紹介する．

2. 先行研究

本章では，使用する表記について説明し，エラーフリー変換アルゴリズムとベクトルの総和と内積に対する高精度計算アルゴリズムの先行研究を紹介する．

まずは本稿で使用する表記について説明する．本研究では IEEE 754 [3] が定める浮動小数点数を扱い，特に明記が無い限りはオーバーフローやアンダーフローを考慮しないこととする．$\mathbb{F}$ $(\subset \mathbb{R})$ をある固定された精度の浮動小数点数の集合とし，$\mathrm{u} = 2^{-p}$ を単位相対丸めとする[*1]．$S_{\min}$ を非正規化数の正の最小数とする[*2]．

浮動小数点演算における表記について説明する．$a, b, c \in \mathbb{F}$ に対して $\mathrm{fl}(a \circ b)$ を浮動小数点演算 $\circ \in \{+, -, \times, \div\}$ の結果とし，$\mathrm{float}(a \circ b \circ c)$ を任意の順序で浮動小数点演算を行った結果とする．本研究における浮動小数点演算の丸めのモードは最近点偶数丸め方式が採用されているとする．

1　芝浦工業大学システム理工学部数理科学科
a)　bv15031@shibaura-it.ac.jp

*1　binary32 では $p = 24$, binary64 では $p = 53$ である．
*2　binary32 では $S_{\min} = 2^{-149}$, binary64 では $S_{\min} = 2^{-1074}$ である．

2.1 エラーフリー変換アルゴリズム

エラーフリー変換アルゴリズムとは，浮動小数点演算における誤差を，演算結果とは別の浮動小数点数で保持することで，情報の損失を無くすアルゴリズムである．和と積に関するエラーフリー変換アルゴリズムをそれぞれ Algorithm 1, Algorithm 2 に示す．なお，本稿で紹介するアルゴリズムは，MATLAB の形式で記述する．Algorithm 2 で使用される Fused Multiply-Add では，$a, b, c \in \mathbb{F}$ に対して $a \cdot b + c \in \mathbb{R}$ の演算を 1 回の丸めで行う．演算結果は $\mathrm{FMA}(a, b, c) \in \mathbb{F}$ と記述する．Fused Multiply-Add が使用できない計算機環境では，文献 [4] のアルゴリズムを用いてエラーフリー変換を行える．

Algorithm 1 浮動小数点演算における和に関するエラーフリー変換アルゴリズム [5]

```
function [x, y] = TwoSum(a, b)
    x = fl(a + b)
    z = fl(x − a)
    y = fl((a − (x − z)) + (b − z))
end function
```

Algorithm 2 浮動小数点演算における積に関するエラーフリー変換アルゴリズム [6]

```
function [x, y] = TwoProdFMA(a, b)
    x = fl(x · y)
    y = FMA(a, b, −x)
end function
```

Algorithm 1 を使用したベクトルの総和に関するエラーフリー変換アルゴリズムを Algorithm 3 に示す．Algorithm 3 では，$p \in \mathbb{F}^n$ に対して $p'_n = \mathrm{fl}(\sum_{i=1}^{n} p_i)$，$\sum_{i=1}^{n} p'_i = \sum_{i=1}^{n} p_i$ となる $p' \in \mathbb{F}^n$ を得る．これは，Kahan [7] によって蒸留アルゴリズム（distillation algorithm）と呼ばれるものである．

Algorithm 3 ベクトルの総和に関するエラーフリー変換アルゴリズム [7]

```
function p′ = VecSum(p)
    for i = 2 : n do
        [p′_i, p′_{i−1}] = TwoSum(p_i, p_{i−1})
    end for
end function
```

2.2 高精度計算アルゴリズム

高精度計算アルゴリズムでは，エラーフリー変換アルゴリズムによって生じた誤差を保持し，最終的な演算結果に反映することで高精度な計算を実現する．ベクトルの内積に関する高精度計算アルゴリズムを Algorithm 4，計算結果に対する誤差上限を定理 1 で紹介する．

Algorithm 4 使用精度の約 2 倍の内部精度でベクトルの内積を計算するアルゴリズム [1]

```
function res = Dot2(x, y)
    [p_1, s_1] = TwoProdFMA(x_1, y_1)
    for i = 2 : n do
        [h_i, r_i] = TwoProdFMA(x_i, y_i)
        [p_i, q_i] = TwoSum(p_{i−1}, h_i)
        s_i = fl(s_{i−1} + (q_i + r_i))
    end for
    res = fl(p_n + s_n)
end function
```

定理 1. $x, y \in \mathbb{F}^n$ に対して上記の Dot2 を実行すると

$$|\mathrm{res} - x^T y| \leq \mathrm{u}|x^T y| + \gamma_n^2 |x^T||y| \qquad (1)$$

が成り立つ．ただし $\gamma_n = n\mathrm{u}/(1 - n\mathrm{u})$, $n\mathrm{u} < 1$ とする．

次に，任意精度におけるベクトルの総和に関する高精度計算アルゴリズムを Algorithm 5 に，計算結果の誤差上限を定理 2 で紹介する．

Algorithm 5 疑似多倍長精度によりベクトルの総和を計算するアルゴリズム [1]

```
function res = SumK(p^(0), K)
    for s = 1 : K − 1 do
        p^(s) = VecSum(p^(s−1))
    end for
    res = fl((Σ_{i=1}^{n−1} p_i^(K−1)) + p_n^(K−1))
end function
```

定理 2. $p^{(0)} \in \mathbb{F}^n$ に対して上記の SumK を実行すると

$$\left|\mathrm{res} - \sum_{i=1}^{n} p_i^{(0)}\right| \leq (\mathrm{u} + 3\gamma_{n-1}^2)\left|\sum_{i=1}^{n} p_i^{(0)}\right| + \gamma_{2n-2}^K \sum_{i=1}^{n} |p_i^{(0)}| \qquad (2)$$

が成り立つ．ただし $4n\mathrm{u} \leq 1$, $K \geq 3$ とする．

Algorithm 5 を使用したベクトルの内積に関する高精度計算アルゴリズムを Algorithm 6，その計算結果の誤差上限を定理 3 で紹介する．

Algorithm 6 疑似多倍長精度によりベクトルの内積を計算するアルゴリズム [1]

```
function res = DotK(x, y, K)
    [p_1, r_1] = TwoProdFMA(x_1, y_1)
    for i = 2 : n do
        [h_i, r_i] = TwoProdFMA(x_i, y_i)
        [p_i, r_{n+i−1}] = TwoSum(p_{i−1}, h_i)
    end for
    r_{2n} = p_n
    res = SumK(r, K − 1)
end function
```

定理 3. $x, y \in \mathbb{F}^n$ に対して上記の DotK を実行すると

$$|\mathrm{res} - x^T y| \le (\mathrm{u} + 2\gamma_{4n-2}^2)|x^T y| + \gamma_{4n-2}^K |x^T||y| \quad (3)$$

が成り立つ．ただし $8n\mathrm{u} \le 1$, $K \ge 3$ とする．

文献 [1] では，浮動小数点演算において誤差上限を返すアルゴリズムの提案もなされている．Algorithm 4 の計算結果に誤差上限を出力に加えたアルゴリズムを Algorithm 7 に示す．Algorithm 7 に関して，定理 4 が成り立つ．

Algorithm 7 使用精度の約 2 倍の内部精度でベクトルの内積を計算する誤差上限付き高精度計算アルゴリズム [1]

```
function [res, err] = Dot2Err(x, y)
    if 2nu ≥ 1, error(‘inclusion failed‘), end
    [p_1, s_1] = TwoProdFMA(x_1, y_1)
    e_1 = |s_1|
    for i = 2 : n do
        [h_i, r_i] = TwoProdFMA(x_i, y_i)
        [p_i, q_i] = TwoSum(p_{i−1}, h_i)
        t_i = fl(q_i + r_i)
        s_i = fl(s_{i−1} + t_i)
        e_i = fl(e_{i−1} + |t_i|)
    end for
    res = fl(p_n + s_n)
    δ = fl((nu)/(1 − 2nu))
    α = fl(u|res| + (δe_n + 3S_min/u))
    err = fl(α/(1 − 2u))
end function
```

定理 4. $x, y \in \mathbb{F}^n$ に対して Dot2Err を実行すると

$$\mathrm{res} - \mathrm{err} \le x^T y \le \mathrm{res} + \mathrm{err} \quad (4)$$

が成り立つ．ただし $2n\mathrm{u} < 1$ とし，アンダーフロー発生時も成り立つ．

3. 提案手法

先行研究に並列計算環境用アルゴリズム PDotK[2] がある．PDotK は逐次実行モデルの高精度計算アルゴリズムを基に設計され，各ノード（または各スレッド）で計算された結果をある 1 ノード（またはある 1 スレッド）に集約し，そこで高精度計算を再度行って最終結果を得る．すなわちマルチノードシングルスレッドまたは，シングルノードマルチスレッドに対応するアルゴリズムである．

分散並列計算環境において並列化効率を上げるためには，MPI（ノード）と OpenMP（スレッド）による階層を持った並列計算を行い，さらにリダクション処理も可能な限り分散させたい．よって本研究では，マルチノードマルチスレッドに対応したアルゴリズムの提案を行う．本章では，先行研究における計算順序に任意性を持たせて分散並列計算環境に対応し，また計算結果の丸め誤差解析を行う．なお，文献 [8], [9], [10] を参考にして誤差解析を行った．以降，定数 $\mathrm{u}' = \mathrm{u}/(1+\mathrm{u})$, $\tilde{\mathrm{u}} = \mathrm{u}'(1+\mathrm{u}')$ を用いる．

3.1 任意の順序で計算可能な高精度計算アルゴリズム

使用精度の約 2 倍の精度でベクトルの総和を任意の順序で計算する高精度計算アルゴリズムを Algorithm 8 に示し，その計算結果の誤差上限を定理 5 に示す．Algorithm 8 では，ベクトルの添え字集合 Λ からある 2 つの要素を選んで計算するため，このアルゴリズムにより全ての計算順序が含まれる．集合 Λ に対して $n(\Lambda)$ は Λ の要素の個数を意味する．

Algorithm 8 使用精度の約 2 倍の精度によるベクトルの総和を計算する，高精度計算かつ任意の順序に対応するアルゴリズム

```
function [p_i, q_i] = Sum2f(p, q)        ▷ If q = NULL then q = 0
    Λ = {1, . . . , n}
    while n(Λ) ≥ 2 do
        i ∈ Λ ∩ ℕ
        j ∈ Λ ∩ ℕ \ i,  Λ = Λ \ j
        [p_i, q̃_i] = TwoSum(p_i, p_j)
        q_i = fl(q_i + (q_j + q̃_i))
    end while
end function
```

定理 5. $p, q \in \mathbb{F}^n$, $q = \mathbf{0}$ に対して上記の Sum2f を実行すると

$$\left|\mathrm{res} - \sum_{i=1}^{n} p_i\right| \le \mathrm{u}' \left|\sum_{i=1}^{n} p_i\right| + (n-1)(n-2)\tilde{\mathrm{u}}\mathrm{u}' \sum_{i=1}^{n} |p_i| \quad (5)$$

が成り立つ [10]．ただし $\mathrm{res} = \mathrm{fl}(p_i + q_i)$, $n \ge 2$ とする．

次に Algorithm 8 を使用して，任意の順序で計算を行っても Algorithm 4 と同程度の結果を得るアルゴリズムを Algorithm 9 に示し，その誤差上限を定理 6 に示す．

Algorithm 9 使用精度の約 2 倍の精度により，ベクトルの内積計算を行う高精度計算かつ任意の順序に対応するアルゴリズム

```
function res = Dot2f(x, y)
    for i = 1 : n do
        [h_i, r_i] = TwoProdFMA(x_i, y_i)
    end for
    [p_n, q_n] = Sum2f(h)
    s_n = float( Σ_{i=1}^{n} r_i )
    res = fl(p_n + (q_n + s_n))
end function
```

定理 6. $x, y \in \mathbb{F}^n$ に対して上記の Dot2f を実行すると

$$|\mathrm{res} - x^T y| \le \mathrm{u}'|x^T y| + (n-1)^2 \tilde{\mathrm{u}}^2 (1+\mathrm{u}')|x^T||y| \quad (6)$$

が成り立つ．

次に Algorithm 3 を任意の順序に拡張したものを Algorithm 10 に示す．

Algorithm 10 任意の順序で計算可能なベクトルの総和に関するエラーフリー変換アルゴリズム

function $p' = \mathrm{VecSumf}(p, m)$
  $\Lambda = \{1, \ldots, m\},\ c = 1$
  while $n(\Lambda) \geq 2$ **do**
    $i \in \Lambda \cap \mathbb{N}$
    $j \in \Lambda \cap \mathbb{N} \setminus i$
    if $i > j$ **then**
      $[p'_i, \tilde{q}_c] = \mathrm{TwoSum}(p_i, p_j),\ \Lambda = \Lambda \setminus j$
    else
      $[p'_j, \tilde{q}_c] = \mathrm{TwoSum}(p_i, p_j),\ \Lambda = \Lambda \setminus i$
    end if
    $c = c + 1$
  end while
  $p'[1, \ldots, m-1] = \tilde{q}[1, \ldots, m-1]$
  if $n > m$ **then**
    $p'[m+1, \ldots, n] = p[m+1, \ldots, n]$
  end if
end function

Algorithm 10 を使用して，Algorithm 5 と同程度の結果を得るアルゴリズムを Algorithm 11 に示し，その誤差上限を定理 7 に示す．

Algorithm 11 任意の順序で計算可能な疑似多倍長精度におけるベクトルの総和に関する高精度計算アルゴリズム

function $\mathrm{res} = \mathrm{SumKf}(p^{(0)}, K)$
  for $s = 1 : K-1$ **do**
    $p^{(s)} = \mathrm{VecSumf}(p^{(s-1)}, n-s+1)$
    $\tilde{p}_s^{(0)} = p_{n-s+1}^{(s)}$
  end for
  $p_{n-K+1}^{(K)} = \mathrm{float}\left(\sum_{i=1}^{n-K+1} p_i^{(K-1)}\right)$
  $\tilde{p}_K^{(0)} = p_{n-K+1}^{(K)}$
  $\mathrm{res} = \mathrm{SumK}(\tilde{p}^{(0)}, K)$
end function

定理 7. $p \in \mathbb{F}^n$ において上記の SumKf を実行すると

$$\left|\mathrm{res} - \sum_{i=1}^{n} p_i^{(0)}\right| \leq \phi_1 \left|\sum_{i=1}^{n} p_i^{(0)}\right| + \phi_2 \sum_{i=1}^{n} |p_i^{(0)}| \tag{7}$$

$$\phi_1 = \mathrm{u}'(1+2(K-1)^2\tilde{\mathrm{u}}),\ \phi_2 = \{(n-1)^K(1+2\mathrm{u}')+(2K)^K\}\mathrm{u}'^K \tag{8}$$

が成り立つ．ただし $1 \geq 4(n-1)\mathrm{u},\ 1 \geq (K-1)(n-1)\mathrm{u},\ 1 \geq 2(K-1)^2\mathrm{u},\ K \geq 3$ とする．

次に Algorithm 11 を使用して，Algorithm 6 と同程度の結果を得るアルゴリズムを Algorithm 12 に示し，その誤差上限を定理 8 に示す．

Algorithm 12 任意の順序で計算可能な疑似多倍長精度におけるベクトルの内積に関する高精度計算アルゴリズム

function $\mathrm{res} = \mathrm{DotKf}(x, y, K)$
  for $i = 1 : n$ **do**
    $[h_i^{(0)}, r_i^{(0)}] = \mathrm{TwoProdFMA}(x_i, y_i)$
  end for
  $\mathrm{res}_h = \mathrm{SumKf}(h^{(0)}, K)$
  $\mathrm{res}_r = \mathrm{SumKf}(r^{(0)}, K-1)$ ▷ If $K-1 = 2$ then Sum2f
  $\mathrm{res} = \mathrm{fl}(\mathrm{res}_h + \mathrm{res}_r)$
end function

定理 8. $x, y \in \mathbb{F}^n$ に対して上記の DotKf を実行すると

$$|\mathrm{res} - x^T y| \leq \phi_3 |x^T y| + \phi_4 |x^T||y| \tag{9}$$

$$\phi_3 = (\mathrm{u} + 2(K\mathrm{u})^2),\ \phi_4 = 2\{(n-1)^K(1+\mathrm{u}') + (2K)^K\}\mathrm{u}^K \tag{10}$$

が成り立つ．ただし $1 \geq 4(n-1)\mathrm{u},\ 1 \geq (K-1)(n-1)\mathrm{u},\ 1 \geq 6(K-1)^2\mathrm{u},\ K \geq 3$ とする．

3.2 任意の順序で計算可能な誤差上限付き高精度計算アルゴリズム

次に任意の順序で計算可能な誤差上限付き高精度計算アルゴリズムの提案を行う．Algorithm 9，Algorithm 11，Algorithm 12 の誤差上限付きアルゴリズムをそれぞれ Algorithm 13，Algorithm 14，Algorithm 15 に示し，その性質をそれぞれ定理 9, 定理 10, 定理 11 として与える．

Algorithm 15 で使用される SumKferrT は，Algorithm 14 内の res と e_3 を返す関数とする．

Algorithm 13 任意の順序で計算可能な使用精度の約 2 倍の精度でベクトルの内積を計算する誤差上限付きアルゴリズム

function $[\mathrm{res}, \mathrm{err}] = \mathrm{Dot2ferr}(x, y)$
  for $i = 1 :\ n$ **do**
    $[h_i, r_i] = \mathrm{TwoProdFMA}(x_i, y_i)$
  end for
  $[p_n, q_n, Q] = \mathrm{Sum2ft}(h)$
  $s_n = \mathrm{float}\left(\sum_{i=1}^{n} r_i\right),\ S = \mathrm{float}\left(\sum_{i=1}^{n} |r_i|\right)$ ▷ 同じ計算順序
  $\mathrm{res} = \mathrm{fl}(p_n + (q_n + s_n))$
  $e = \mathrm{fl}(Q + S)$
  $\psi = \mathrm{fl}\left(\frac{n\mathrm{u}}{1-n\mathrm{u}}\right)$
  $\alpha = \mathrm{fl}((\mathrm{u} \cdot |\mathrm{res}| + \psi \cdot e) + 3S_{\min}/\mathrm{u})$
  $\mathrm{err} = \mathrm{fl}(\alpha/(1-4\mathrm{u}))$
end function

定理 9. $x, y \in \mathbb{F}^n$ に対して Dot2ferr を実行すると

$$\mathrm{res} - \mathrm{err} \leq x^T y \leq \mathrm{res} + \mathrm{err} \tag{11}$$

が成り立つ．ただし $n\mathrm{u} < 1$ とし，アンダーフロー発生時も成り立つ．

Algorithm 14 任意の順序で計算可能な疑似多倍長精度によりベクトルの総和を計算する誤差上限付きアルゴリズム

function $[\mathrm{res}, \mathrm{err}] = \mathrm{SumKferr}(p^{(0)}, K)$
  for $s = 1 : K-1$ **do**
    $p^{(s)} = \mathrm{VecSumf}(p^{(s-1)}, n-s+1)$
    $\tilde{p}_s^{(0)} = p_{n-s+1}^{(s)}$
  end for
  $p_{n-K+1}^{(K)} = \mathrm{float}\left(\sum_{i=1}^{n-K+1} p_i^{(K-1)}\right)$
  $Q = \mathrm{float}\left(\sum_{i=1}^{n-K+1} |p_i^{(K-1)}|\right)$ ▷ 上記と同じ計算順序
  $\tilde{p}_K^{(0)} = p_{n-K+1}^{(K)}$
  for $s = 1 : K-1$ **do**
    $\tilde{p}^{(s)} = \mathrm{VecSum}(\tilde{p}^{(s-1)})$
  end for
  $\mathrm{res} = \mathrm{fl}\left(\sum_{i=1}^{K} \tilde{p}_i^{(K-1)}\right)$
  $S = \mathrm{fl}\left(\sum_{i=1}^{K} |\tilde{p}_i^{(K-1)}|\right)$ ▷ 上記と同じ計算順序
  $\psi_1 = \mathrm{fl}\left(\frac{(n-K)\mathrm{u}}{1-(n-K)\mathrm{u}}\right)$, $\psi_2 = \mathrm{fl}\left(\frac{(K-1)\mathrm{u}}{1-(K-1)\mathrm{u}}\right)$
  $e_1 = \mathrm{fl}(\psi_1 Q)$, $e_2 = \mathrm{fl}(\psi_2 S)$, $e_3 = \mathrm{fl}(e_1 + e_2)$
  $\alpha = \mathrm{fl}(e_3 + 2S_{\min})$
  $\mathrm{err} = \mathrm{fl}(\alpha/(1-3\mathrm{u}))$
end function

定理 10. $p^{(0)} \in \mathbb{F}^n$ に対して SumKferr を実行すると

$$\mathrm{res} - \mathrm{err} \le \sum_{i=1}^{n} p^{(0)} \le \mathrm{res} + \mathrm{err} \qquad (12)$$

が成り立つ．ただし $1 \ge 4(n-1)\mathrm{u}$, $1 \ge (K-1)(n-1)\mathrm{u}$, $1 \ge 2(K-1)^2\mathrm{u}$, $K \ge 3$ とし，アンダーフロー発生時も成り立つ．

Algorithm 15 任意の順序で計算可能な疑似多倍長精度におけるベクトルの内積に関する誤差上限付き高精度計算アルゴリズム

function $[\mathrm{res}, \mathrm{err}] = \mathrm{DotKferr}(x, y, K)$
  for $i = 1 : n$ **do**
    $[h_i^{(0)}, r_i^{(0)}] = \mathrm{TwoProdFMA}(x_i, y_i)$
  end for
  $[\mathrm{res}_h, e_h] = \mathrm{SumKferrT}(h^{(0)}, K)$
  $[\mathrm{res}_r, e_r] = \mathrm{SumKferrT}(r^{(0)}, K-1)$
  $[\mathrm{res}, \mathrm{res}'] = \mathrm{TwoSum}(\mathrm{res}_h, \mathrm{res}_r)$
  $\alpha = \mathrm{fl}(|\mathrm{res}'| + (e_h + e_r) + 3S_{\min}/\mathrm{u})$
  $\mathrm{err} = \mathrm{fl}(\alpha/(1-5\mathrm{u}))$
end function

定理 11. $x, y \in \mathbb{F}^n$ に対して DotKferr を実行すると

$$\mathrm{res} - \mathrm{err} \le x^T y \le \mathrm{res} + \mathrm{err} \qquad (13)$$

が成り立つ．ただし $1 \ge 4(n-1)\mathrm{u}$, $1 \ge (K-1)(n-1)\mathrm{u}$, $1 \ge 6(K-1)^2\mathrm{u}$, $K \ge 3$ とし，アンダーフロー発生時も成り立つ．

4. 実装について

使用精度を倍精度浮動小数点数とし，疑似 4 倍精度により行列･ベクトル積を計算するルーチン PDDot2MV, 疑似多倍長精度行列・ベクトル積ルーチン PDDotKMV を開発した．また，それぞれの誤差上限付きルーチン PDDot2ErrMV, PDDotKErrMV も併せて開発した．PBLAS の行列・ベクトル積ルーチン PDGEMV と同様の仕様にするため，

- 行列とベクトルの分散情報 (descriptor)
- 転置オプション
- 定数倍オプション
- 先頭ポインタの指定オプション
- インクリメント幅の指定オプション

を設定できる．

4.1 PDDot2MV の実装

PDot2MV の実装の基本となるアルゴリズムを Algorithm 16 に示す．Algorithm 16 は，Algorithm 9 に包括されるため定理 6 が成り立つ．以下，n' はノード内の各スレッドが担当するベクトルのサイズとする．

Algorithm 16 PDot2f

function $\mathrm{res} = \mathrm{PDot2f}(x, y)$
  %begin MPI M nodes parallel%
  %begin OMP m threads parallel%
  $[\tilde{p}_1, \tilde{r}_1] = \mathrm{TwoProdFMA}(\tilde{x}_1, \tilde{y}_1)$
  for $i = 2 : n'$ **do**
    $[\tilde{h}_i, \tilde{r}_i] = \mathrm{TwoProdFMA}(\tilde{x}_i, \tilde{y}_i)$
    $\tilde{s}_i = \mathrm{fl}(\tilde{r}_i + \tilde{r}_{i-1})$
    $[\tilde{p}_i, \tilde{q}_i] = \mathrm{TwoSum}(\tilde{p}_{i-1}, \tilde{h}_i)$
    $\tilde{q}_i = \mathrm{fl}(\tilde{q}_i + \tilde{q}_{i-1})$
  end for
  $\hat{p}_{id} = \tilde{p}_{n'}$, $\hat{s}_{id} = \tilde{s}_{n'}$, $\hat{q}_{id} = \tilde{q}_{n'}$
  %end OMP m threads parallel%
  for $id = 2 : m$ **do**
    $\hat{s}_{id} = \mathrm{fl}(\hat{s}_{id} + \hat{s}_{id-1})$
    $[\hat{p}_{id}, \check{q}_{id}] = \mathrm{TwoSum}(\hat{p}_{id}, \hat{p}_{id-1})$
    $\hat{q}_{id} = \mathrm{fl}(\hat{q}_{id} + (\hat{q}_{id-1} + \check{q}_{id}))$
  end for
  $p_{Id} = \hat{p}_m$, $s_{Id} = \hat{s}_m$, $q_{Id} = \hat{q}_m$
  %begin MPI Communication%
  $[p_1, q_1] = \mathrm{Sum2f}(p, q)$
  $s_1 = \mathrm{float}\left(\sum_{i=1}^{M} s_i\right)$
  %end MPI Communication%
  %end MPI M nodes parallel%
  $\mathrm{res} = \mathrm{fl}(p_1 + (q_1 + s_1))$
end function

4.2 PDDotKMV の実装

PDDotKMV の実装は，Algorithm 12 に包括されるアルゴリズム設計で行った．Algorithm 12 内の Algorithm 2 を用いて行われるベクトル要素積の処理は，独立した計算のため，並列化が可能である．また Algorithm 11 が並列実

行可能であれば，PDDotKMV の実装が可能となる．

Algorithm 11 を 3 ノード，$K = 3$ で並列実行時した際の実行モデルを図 1 に示す．図 1 のような実装を行えば，Algorithm 11 が並列実行可能となる．

ノード 1	ノード 2	ノード 3
$\cdots p^{(0)}_{n_1-2}, p^{(0)}_{n_1-1}, p^{(0)}_{n_1}$	$\cdots p^{(0)}_{n_2-2}, p^{(0)}_{n_2-1}, p^{(0)}_{n_2}$	$\cdots p^{(0)}_{n_3-2}, p^{(0)}_{n_3-1}, p^{(0)}_{n_3}$

（大域ベクトル p をノードごとに分割）

⇓

VecSumf を 2 回繰り返し実行（ノード内）

$\cdots p^{(2)}_{n_1-2}, p^{(2)}_{n_1-1}, p^{(1)}_{n_1}$	$\cdots p^{(2)}_{n_2-2}, p^{(2)}_{n_2-1}, p^{(1)}_{n_2}$	$\cdots p^{(2)}_{n_3-2}, p^{(2)}_{n_3-1}, p^{(1)}_{n_3}$

$(p^{(1)}_{n_*} = \mathrm{float}(\sum_{i=1}^{n_*} p_i^{(0)})$，$p^{(2)}_{n_*-1} = \mathrm{float}(\sum_{i=1}^{n_*-1} p_i^{(1)}))$

⇓

誤差要素のリダクション処理（ノード内）

$p^{(3)}_{n_1-2}, p^{(2)}_{n_1-1}, p^{(1)}_{n_1}$	$p^{(3)}_{n_2-2}, p^{(2)}_{n_2-1}, p^{(1)}_{n_2}$	$p^{(3)}_{n_3-2}, p^{(2)}_{n_3-1}, p^{(1)}_{n_3}$

$(p^{(3)}_{n_*-2} = \mathrm{float}(\sum_{i=1}^{n_*-2} p_i^{(2)}))$

⇓

1 回目の VecSumf を実行（グローバル）

$p^{(3)}_{n_1-2}, p^{(2)}_{n_1-1}, p^{(2)}_{n_1}$	$p^{(3)}_{n_2-2}, p^{(2)}_{n_2-1}, p^{(2)}_{n_2}$	$p^{(3)}_{n_3-2}, p^{(2)}_{n_3-1}, p^{(1)}_{n_3}$

$(p^{(1)}_{n_3} = \mathrm{float}(\sum p))$

⇓

2 回目の VecSumf を実行（グローバル）

$p^{(3)}_{n_1-2}, p^{(3)}_{n_1-1}, p^{(3)}_{n_1}$	$p^{(3)}_{n_2-2}, p^{(3)}_{n_2-1}, p^{(3)}_{n_2}$	$p^{(3)}_{n_3-2}, p^{(2)}_{n_3-1}, p^{(1)}_{n_3}$

⇓

誤差要素のリダクション処理（グローバル）

		$p^{(3)}_{n_3-2}, p^{(2)}_{n_3-1}, p^{(1)}_{n_3}$

⇓

$p^{(3)}_{n_3-2}, p^{(2)}_{n_3-1}, p^{(1)}_{n_3}$ に対して SumK を実行し計算結果を得る．

図 1 SumKf の並列実行モデル

4.3 リダクション処理の効率化

分散並列計算では，ベクトルの総和に関するリダクション処理をノード間で行う必要がある．逐次実行モデルを基にしたアルゴリズムでノード間のリダクションを行う際は，各ノードで処理を行った結果をある 1 つのノードにまとめて，リダクションを行っていた．しかし，先の処理では他のノードが待ち状態となり処理の分散効率が良くない．

本稿で提案したアルゴリズムでは，任意の順序で計算可能なため，ノード間でリダクション処理を行う際にペアノード間通信で行っても差し支えない．よって提案ルーチンでは図 2 のようなノード間通信を採用した．

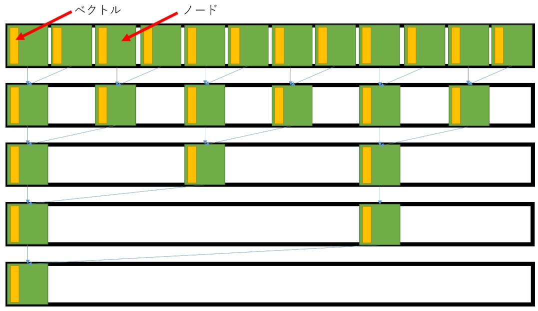

図 2 ペアノード間通信によるリダクション処理

4.4 ホットスポットの特定による高速化

提案ルーチンの高速化のため，ホットスポットの特定を行った．PDDot2MV に対して調査を行ったところ，各 MPI ノードの各 OpenMP スレッドが行う Algorithm 1，Algorithm 2 の処理がホットスポットであった．

よってホットスポットに対し，インライン展開とループアンローリングを施し，高速化を達成した．1 ノード当たりの行列サイズが 10000 の正方行列とベクトルの積を 4×4 ノードで実施した結果，表 1 を得た．理化学研究所の京コンピュータでは約 3.4 倍，名古屋大学にある富士通 FX100 では約 6.3 倍の高速化に成功した．

表 1 インライン展開とループアンローリングによる実行時間比

インライン展開	無	有	有	有	有	有
展開数		1	2	4	8	16
京コンピュータ	1.00	0.71	0.53	0.37	0.30	0.29
FX100	1.00	0.89	0.51	0.30	0.19	0.16

5. 数値実験結果

本章では，理化学研究所の京コンピュータと名古屋大学の FX100 を使用して数値実験を行った結果を紹介する．

5.1 実行時間について

PBLAS の行列ベクトル積ルーチン PDGEMV に対する提案ルーチン（PDDot2MV，PDotKMV）の実行時間比の調査を行った．図 3，図 4 に京コンピュータと FX100 のそれぞれの結果を示す．また図 5，図 6 に転置オプションありの場合を示す．ただし縦軸を実行時間比率，横軸を疑似多倍長精度（K 倍精度）の K[*3] とする．実験環境は $n \times n$ の平方ノードで行い，1 ノード当たりの行列サイズを 10000 とする．

転置オプションなしのとき，$K = 2$ における実行時間比は，京コンピュータでは 10〜15 倍程度であり，FX100 では 4〜6 倍程度であった．$K = 3$ における実行時間比は，京コンピュータでは 75 倍程度であり，FX100 では 65 倍程

[*3] $K = 2$ では PDDot2MV，$K = 3$ 以降では PDDotKMV を使用する．

度であった．また $K = 4, 5, 6, 7, 8, 9$ における実行時間比は，京コンピュータならびに FX100 で，K に比例的な増加が確認された．

転置オプションありのとき，$K = 2$ における実行時間比は，京コンピュータでは 18〜36 倍程度であった．ノード数増加に伴い，実行時間比が増加傾向にあった．FX100 では 12〜13 倍程度であった．$K = 3$ における実行時間比は，京コンピュータでは 52〜61 倍程度であり，FX100 では 47 倍程度であった．また転置オプションなしのとき同様に，K に比例的に実行時間比が増加傾向にあった．

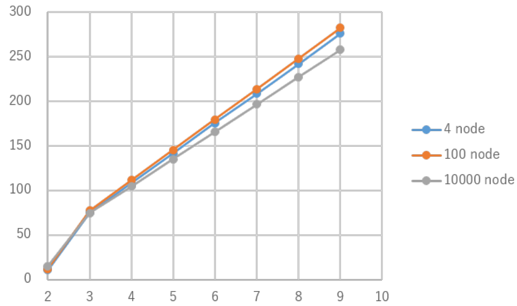

図 3 PDDot2MV，PDDotKMV の実行時間比（京，転置オプションなし）

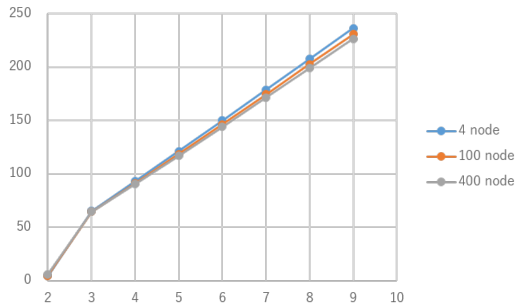

図 4 PDDot2MV，PDDotKMV の実行時間比（FX100，転置オプションなし）

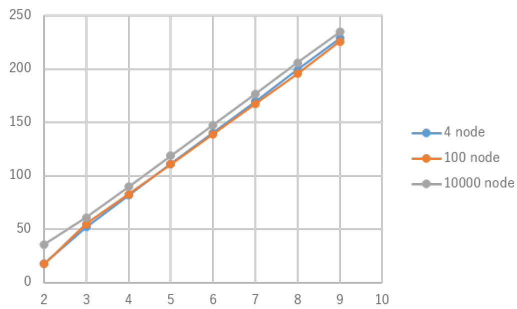

図 5 PDDot2MV，PDDotKMV の実行時間比（京，転置オプションあり）

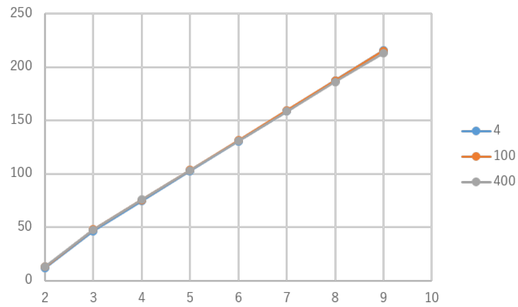

図 6 PDDot2MV，PDDotKMV の実行時間比（FX100，転置オプションあり）

次に提案ルーチン（PDDot2MV，PDotKMV）の FLOPS を図 7，図 8 に示す．なお $K = 1$ は PBLAS の行列・ベクトル積ルーチン PDGEMV とする．

FX100 では，PDGEMV よりも PDDot2MV のほうがパフォーマンスが良いという結果を得た．

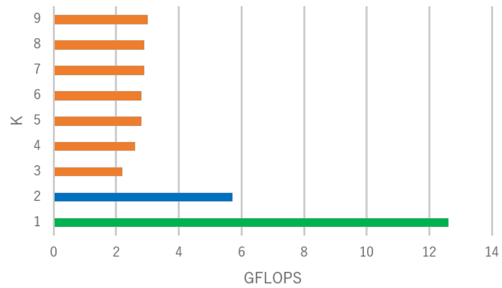

図 7 FLOPS の比較（京）

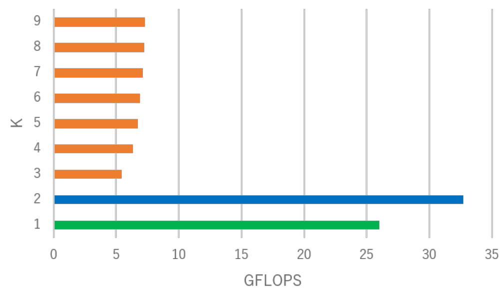

図 8 FLOPS の比較（FX100）

次に提案ルーチン（PDDot2MV，PDotKMV）に対する誤差上限付き提案ルーチン（PDDot2ErrMV，PDDotKErrMV）の実行時間比の調査を行った．図 9，図 10 に京コンピュータと FX100 のそれぞれの結果を示す．ただし縦軸を実行時間比率，横軸を疑似多倍長精度（K 倍精度）の K とする．実験環境は $n \times n$ の平方ノードで行い，1 ノード当たりの行列サイズを 10000 とする．

$K = 2$ における実行時間比は，京コンピュータならびに FX100 で 1.1 倍〜1.4 倍程度であった．$K = 3$ における実

行時間比は，京コンピュータならびに FX100 で 1.1 倍〜1.2 倍程度であった．また K の増加に伴い，実行時間比が減少傾向にあった．

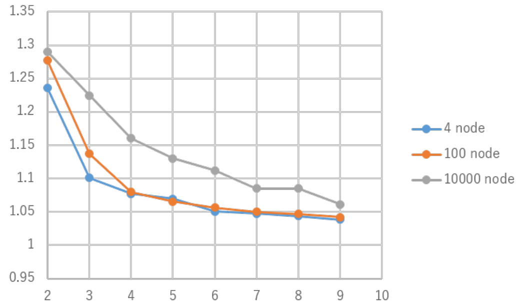

図 9 PDDot2ErrMV，PDDotKErrMV の実行時間比（京）

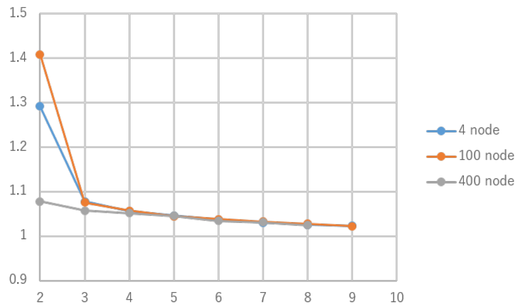

図 10 PDDot2ErrMV，PDDotKErrMV の実行時間比（FX100）

5.2 精度について

$x, y \in \mathbb{R}^n$ に対して

$$\mathrm{cond}(x^T y) = 2\frac{|x^T||y|}{|x^T y|} \tag{14}$$

は内積の条件数である [1]．誤差上限付き提案ルーチンで得られる真値との相対誤差を内積の条件数（$1 \leq s \leq 7$，u^{-s} 程度[*4]）に応じて調査した．図 11 に京コンピュータで得られた結果を示す．ただし相対誤差が 1.0 を超えたときはプロットしない．実験環境は 2×2 の平方ノードで行い，1 ノード当たりの行列サイズを 10000 とする．

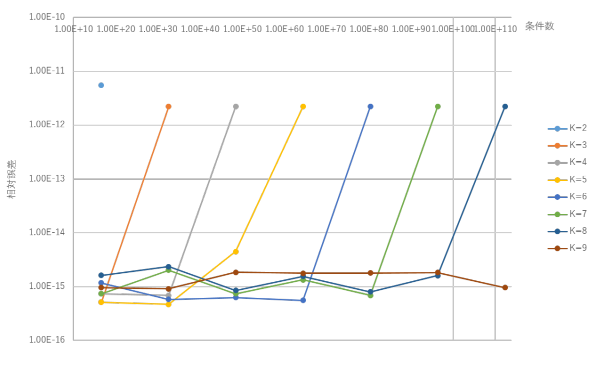

図 11 誤差上限から得られる相対誤差と内積条件数の関係（京）

[*4] 今回の実験では binary64 で行ったため，$\mathrm{u}^{-s} = 2^{53s}$ である．

京コンピュータならびに FX100 で同程度の結果を得た．条件数が u^{-s} 程度のとき $K = s + 2$ 以上の精度であれば，誤差上限付き提案ルーチンで得られる誤差上限で，演算結果が真値に十分近い値だと保証されることを確認した．

6. まとめ

本研究では，任意の順序で計算を行っても誤差評価が可能な高精度計算アルゴリズムの提案を行った．また分散並列計算環境への応用として，行列・ベクトル積ルーチンの実装を行い，性能評価を実施した．

京コンピュータにおける性能評価において，提案ルーチン PDDot2MV では，PBLAS の行列・ベクトル積ルーチン PDGEMV の 10〜15 倍程度である．計算回数は理論値で 10 倍程度であることを考慮すれば，提案ルーチン PDDot2MV は有用であると言える．

今後は，提案ルーチン PDDotKMV の高速化や分散並列計算環境用アルゴリズムの誤差解析に注力したい．

謝辞 本研究は，文部科学省ポスト「京」萌芽的課題 1「基礎科学のフロンティア - 極限への挑戦（極限の探究に資する精度保証付き数値計算学の展開と超高性能計算環境の創成）」の一環として実施した．また理化学研究所の京コンピュータならびに名古屋大学情報基盤センターの FX100 を使用して実験結果を得た．

参考文献

[1] Ogita T., Oishi S., and Rump S. M.: Accurate Floating-point Sum And Dot Product, SIAM Journal on Scientific Computing, 26:6 (2005), 1955–1988.
[2] Ogita T., Oishi S., Rump S. M. and Yamanaka N.: A Parallel Algorithm of Accurate Dot Product, Parallel Computing, 34:6-8 (2008), 392–410.
[3] IEEE Computer Society, IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, (2008).
[4] Dekker T. J.: A floating-point technique for extending the available precision, Numerische Mathematik, 18:3 (1971), 224-242.
[5] Knuth D. E.: The Art of Computer Programing volume 2, Addison Wesley, (1998).
[6] Nievergelt Y.: Scalar Fused Multiply-Add Instructions Produce Floating-Point Matrix Arithmetic Provably Accurate to the Penultimate Digit, ACM Trans. Math. Softw., 29:1 (2003), 27–48.
[7] Kahan W.: Doubled-precision IEEE standard 754 floating-point arithmetic, Unpublished manuscript, (1987).
[8] Higham, N. J.: Accuracy and stability of numerical algorithms, Siam, (2002).
[9] Jeannerod C. P. and Rump S. M.: Improved error bounds for inner products in floating-point arithmetic, SIAM Journal on Matrix Analysis and Applications, 34:2 (2013), 338–344.
[10] Jeannerod C. P. and Rump S. M.: On relative errors of floating-point operations: Optimal bounds and applications, Mathematics of Computation, 87:310 (2018), 803–819.