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　61–72 (June 2008)

行列積を用いた古典 Gram-Schmidt 直交化法の並列化

横澤 拓弥[†1]　高橋 大介[†1]
朴 泰祐[†1]　佐藤 三久[†1]

本論文では，直交化アルゴリズムの 1 つである古典 Gram-Schmidt 法（CGS 法）の効率的な実装を行い，並列化して評価した結果について述べる．CGS 法においては，内積計算とベクトル変換を行列積に変更することで高速化できることが知られている．本論文では，CGS 法を行列積で行う手法を拡張し，行列積を適用できる範囲の比率を高めることで性能を改善することができることを示す．また，行列積の実装上の特徴から，最適な分割手法が存在することを示す．提案する手法を PC クラスタに実装し，性能評価を行った．その結果，32 ノードの Xeon 3 GHz PC クラスタにおいて，naive な実装に対し約 5.36 倍の高速化となり，約 122.9 GFLOPS の性能を得ることができた．

Efficient Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication

Takuya Yokozawa,[†1] Daisuke Takahashi,[†1]
Taisuke Boku[†1] and Mitsuhisa Sato[†1]

In this paper, we propose an efficient parallel implementation of classical Gram-Schmidt (CGS) orthogonalization. It is known that the CGS orthogonalization of a matrix can be altered into a matrix multiplication. We show that the CGS orthogonalization with matrix multiplication improves performance effectively and blocking method using square matrix multiplication improve performance because of matrix's row-column ratio affects DGEMM routine performance. We succeeded in obtaining performance of approximately 122.9 GFLOPS on a 32-node Xeon 3 GHz PC cluster.

1. はじめに

直交化処理は，複数の一次独立なベクトルの組を正規直交基底に変換するものであり，線形計算において基本的かつ重要な処理の 1 つである．Gram-Schmidt 直交化は，直交化処理を行う手法として広く知られており，固有値問題や連立一次方程式の反復法における Krylov 部分空間の計算などにおいて広く用いられている手法である．

科学技術計算においては，Gram-Schmidt の直交化を用いて行列の対角化を行うことが多いが，この場合 $N \times N$ 行列に対して $O(N^3)$ の計算量を必要とすることから，Gram-Schmidt の直交化を高速に，かつ高精度に求めることは重要であり，これまでに多くのアルゴリズムが提案されている[1)]．

Gram-Schmidt の直交化としては，古典的な計算順序に従う方法（Classical Gram-Schmidt 法，以下 CGS 法）および，計算誤差の蓄積を考慮した計算順序に従う方法（Modified Gram-Schmidt 法，以下 MGS 法）が知られている．

複数のベクトルに対して行う CGS 法においては，複数のベクトルに対する内積および複数のベクトルに対するベクトル変換をレベル 3 BLAS である行列積に変換することでさらに高速化できることが知られている[2)]．また，CGS 法を行列積に変換して高速化する手法において，すでにいくつかの手法について評価がなされている[3)]．なお，MGS 法を用いることで，CGS 法における精度の問題を解決することができるが，計算順序の依存性により，レベル 2 BLAS である行列ベクトル積を用いる必要があるため，高速化に向いていない．また，要求精度を満たすまで CGS 法を繰り返す Daniel-Gragg-Kaufman-Stewart 型 Gram-Schmidt 法（DGKS 法）[4)] や Iterated-CGS 法（ICGS 法）[5)] を用いることで，MGS 法よりも精度を良くすることができることが知られている[5),6)]．したがって本論文では，CGS 法を行列積で行う手法について考察し，CGS 法における行列積の最適化手法を提案する．

以下，2 章で古典 Gram-Schmidt 直交化（CGS）法について説明する．3 章で提案手法について述べ，4 章で，その並列化について述べる．5 章では，提案手法の計算量および通信量について考察し，6 章で性能評価の結果を示す．7 章はまとめである．

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

2. 古典 Gram-Schmidt 直交化法

本章では，古典 Gram-Schmidt 直交化法（以下 CGS 法）について説明する．

m 行 n 列の行列 $A = (\boldsymbol{a_1 a_2} \cdots \boldsymbol{a_n})$ に対する CGS 法は，**図 1** のように記述される．なお，$(\boldsymbol{q_i}, \boldsymbol{a_j})$ はベクトル $\boldsymbol{q_i}$ と $\boldsymbol{a_j}$ の内積を表し，$||\boldsymbol{q_j}||$ は，ベクトル$\boldsymbol{q_j}$のユークリッドノルムを表している．

図 1 で示した CGS 法においては，$\boldsymbol{q_j}$ の計算をする内側のループに対してレベル 2 BLAS である行列ベクトル積を用いることができるため，内積計算の高速化が期待される．

2.1 行列積を用いた CGS 法

図 1 で示した CGS 法において，複数のベクトル $\boldsymbol{q_1}, \boldsymbol{q_2}, \cdots, \boldsymbol{q_{j-1}}$ の直交化がすでに済んでいれば，$\boldsymbol{a_j}, \boldsymbol{a_{j+1}}, \cdots, \boldsymbol{a_n}$ との内積計算には依存性がないため，$\boldsymbol{q_1}, \boldsymbol{q_2}, \cdots, \boldsymbol{q_n}$ の計算において，前半の内積計算と後半のベクトル変換を独立に計算することができることが知られている[2)]．

図 1 において $n = 6$ とした場合，$\boldsymbol{q_1}$〜$\boldsymbol{q_6}$ を計算すると以下のようになる．

$$\boldsymbol{q_1} = \boldsymbol{a_1}, \quad \boldsymbol{q_1} = \frac{\boldsymbol{q_1}}{||\boldsymbol{q_1}||} \tag{1}$$

$$\boldsymbol{q_2} = \boldsymbol{a_2} - \sum_{i=1}^{1} (\boldsymbol{q_i}, \boldsymbol{a_2})\boldsymbol{q_i}, \quad \boldsymbol{q_2} = \frac{\boldsymbol{q_2}}{||\boldsymbol{q_2}||} \tag{2}$$

$$\boldsymbol{q_3} = \boldsymbol{a_3} - \sum_{i=1}^{2} (\boldsymbol{q_i}, \boldsymbol{a_3})\boldsymbol{q_i}, \quad \boldsymbol{q_3} = \frac{\boldsymbol{q_3}}{||\boldsymbol{q_3}||} \tag{3}$$

$$\boldsymbol{q_4} = \boldsymbol{a_4} - \sum_{i=1}^{3} (\boldsymbol{q_i}, \boldsymbol{a_4})\boldsymbol{q_i}, \quad \boldsymbol{q_4} = \frac{\boldsymbol{q_4}}{||\boldsymbol{q_4}||} \tag{4}$$

$$\boldsymbol{q_5} = \boldsymbol{a_5} - \sum_{i=1}^{4} (\boldsymbol{q_i}, \boldsymbol{a_5})\boldsymbol{q_i}, \quad \boldsymbol{q_5} = \frac{\boldsymbol{q_5}}{||\boldsymbol{q_5}||} \tag{5}$$

$$\boldsymbol{q_6} = \boldsymbol{a_6} - \sum_{i=1}^{5} (\boldsymbol{q_i}, \boldsymbol{a_6})\boldsymbol{q_i}, \quad \boldsymbol{q_6} = \frac{\boldsymbol{q_6}}{||\boldsymbol{q_6}||} \tag{6}$$

```
do j = 1, n
  q_j = a_j
  do i = 1, j − 1
    q_j = q_j − (q_i, a_j) q_i
  end do
  q_j = q_j / ||q_j||
end do
```

図 1　古典 Gram-Schmidt 直交化法
Fig. 1　Classical Gram-Schmidt orthogonalization algorithm.

ところが，式 (4)〜(6) で $\boldsymbol{q_4}$，$\boldsymbol{q_5}$，$\boldsymbol{q_6}$ を計算する際に，$\boldsymbol{q_1}$，$\boldsymbol{q_2}$，$\boldsymbol{q_3}$ の直交化がすでに済んでいれば，式 (7)，(8) によって依存性のない部分を計算し，その後に $\boldsymbol{q_4}$ に対して依存性のある $\boldsymbol{q_5}$，$\boldsymbol{q_6}$ を式 (9)，(10) によって計算することができる．

$$S = Q_{13}^t A_{13} \tag{7}$$

$$Q_{46} = A_{46} - Q_{13} S \tag{8}$$

$$\boldsymbol{q_5} = \hat{\boldsymbol{q_5}} - (\boldsymbol{q_4}, \boldsymbol{a_5})\boldsymbol{q_4}, \quad \boldsymbol{q_5} = \frac{\boldsymbol{q_5}}{||\boldsymbol{q_5}||} \tag{9}$$

$$\boldsymbol{q_6} = \hat{\boldsymbol{q_6}} - (\boldsymbol{q_4}, \boldsymbol{a_6})\boldsymbol{q_4} - (\boldsymbol{q_5}, \boldsymbol{a_6})\boldsymbol{q_5}, \quad \boldsymbol{q_6} = \frac{\boldsymbol{q_6}}{||\boldsymbol{q_6}||} \tag{10}$$

ここで，$Q_{1,3} = (\boldsymbol{q_1 q_2 q_3})$，$Q_{4,6} = (\boldsymbol{q_4 \hat{q}_5 \hat{q}_6})$ であり，$A_{1,3} = (\boldsymbol{a_1 a_2 a_3})$，$A_{4,6} = (\boldsymbol{a_4 a_5 a_6})$ である．また，$\hat{\boldsymbol{q_5}}$ および $\hat{\boldsymbol{q_6}}$ は式 (8) によって部分的に求まっている $\boldsymbol{q_5}$ および $\boldsymbol{q_6}$ の値である．

上記の変換によって，$\boldsymbol{q_4}$，$\boldsymbol{q_5}$，$\boldsymbol{q_6}$ の計算において，6 回のベクトル行列積が，2 回の行列積と 4 回のベクトル行列積に変換できることが分かる．

文献 2) に示されている手法を $N \times N$ 行列 $A = (\boldsymbol{a_1 a_2} \cdots \boldsymbol{a_N})$ に適用し，$(N/2) \times (N/2)$ の正方行列に対する行列積を用いた例を**図 2** に示す．図 2 で示されている三角形において，縦方向は直交化するベクトルの本数を表し，横方向はベクトルの直交化に必要な項数を表している．

図 2 においては，II で示されている範囲を $(N/2) \times (N/2)$ の正方行列に対する行列積として計算するとともに，I および II で示されている範囲を，それぞれ N 回の行列ベクトル積として計算している．つまり，II で示されている範囲はレベル 3 BLAS である行列積

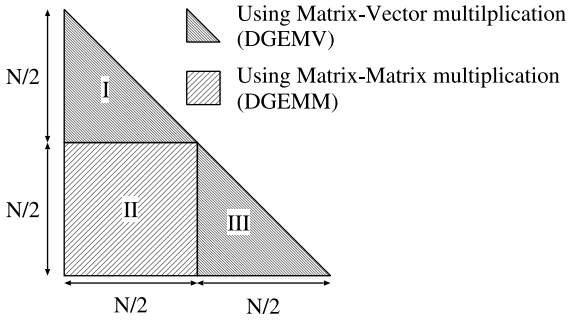

図 2　従来の手法における計算空間の分割方法
Fig. 2　Computational space blocking algorithm of conventional method.

（GEMM）で計算することができるが，I および III で示されている範囲はレベル 2 BLAS アルゴリズムである行列ベクトル積（GEMV）で計算することになり，行列積で計算できるのは全演算量の半分であることが分かる．

したがって，レベル 3 BLAS である行列積で計算できる部分を増やすことができれば，さらにキャッシュの再利用を図ることができ，演算性能を向上させることが可能であると考えられる．

図 3 に示すように，DGEMM と DGEMV の性能差が小さいため，DGEMM の適用サイズには下限があることが予想される．

2.2　CGS 法の分割

CGS 法を行列ベクトル積を用いる部分と行列積を用いる部分とに分割できることを前節で示したが，本節ではそれぞれの部分について説明する．

2.2.1　行列ベクトル積を用いる計算部分

$N \times N$ 行列 A に対する CGS 法による直交化の計算において，$s-1$ 本のベクトル $\boldsymbol{a_1}, \cdots, \boldsymbol{a_{s-1}}$ がすでに直交化されているとき，h 本のベクトル $\boldsymbol{a_s}, \cdots, \boldsymbol{a_{s+h-1}}$ の直交化の計算において行列ベクトル積を用いる部分は，**図 4** のように書くことができる．

以降，行列ベクトル積（GEMV）を用いた計算部分を TRI 関数とする．ここで，A および Q は，$N \times N$ 行列であり，それぞれ直交化の対象の行列と計算途中の値を含んだ直交化後の行列である．s は計算を開始するベクトルの位置であり $\boldsymbol{q_1}, \cdots, \boldsymbol{q_{s-1}}$ はすでに直交化されているものとする．h は計算対象のベクトルの本数であり，計算空間の三角形の高さ（height）に相当する．$Q_{i,j}$ は $N \times N$ 行列 $Q = (\boldsymbol{q_1 q_2} \cdots \boldsymbol{q_N})$ における $\boldsymbol{q_i}$ から $\boldsymbol{q_j}$ を表している．

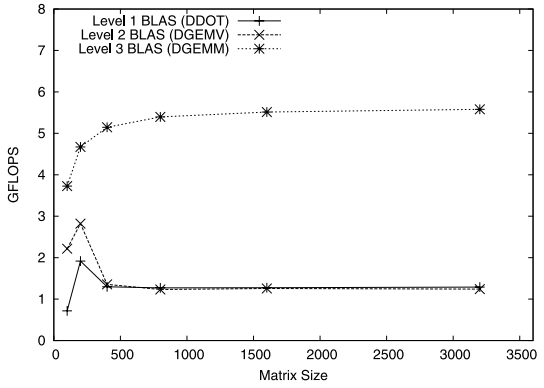

図 3　レベル 1 BLAS（DDOT），レベル 2 BLAS（DGEMV），レベル 3 BLAS（DGEMM）を用いた行列積の性能比較（Pentium 4 3.4 GHz，1CPU）
Fig. 3　Matirx muliplication performance comparison of using level 1 BLAS (DDOT), level 2 BLAS (DGEMV) and level 3 BLAS (DGEMM) on Pentium 4 3.4 GHz (1CPU).

```
begin TRI(A, Q, N, s, h)
  q_s = q_s/||q_s||
  do i = s + 1, s + h
    w = Q_{s,i}^T a_i
    q_i = q_i − Q_{s,i} w
    q_i = q_i/||q_i||
  end do
end
```

図 4　行列ベクトル積を用いる計算部分（TRI 関数）
Fig. 4　Part of matrix vector multiplication (TRI function).

2.2.2　行列積を用いる計算部分

$N \times N$ 行列 A および Q に対する CGS 法による直交化の計算において，$s-1$ 本のベクトル $\boldsymbol{a_1}, \cdots, \boldsymbol{a_{s-1}}$ がすでに直交化されているとき，h 本のベクトル $\boldsymbol{a_s}, \cdots, \boldsymbol{a_{s+h-1}}$ の直交化の計算のうち，w 本の直交化済みのベクトル $\boldsymbol{q_1}, \cdots, \boldsymbol{q_w}$ を用いて行列積によって計算する部分を，**図 5** のように書くことができる．

```
begin RECT(A, Q, N, s, w, h)
  S = Q^t_{s, s+h/2-1} A_{s, s+h/2-1}
  Q_{s+h/2, s+h-1} = Q_{s+h/2, s+h-1} − Q_{s, s+h/2-1} S
end
```

図 5　行列積を用いる計算部分（RECT 関数）
Fig. 5　Part of matrix multiplication (RECT function).

以降，行列積を用いた部分を RECT 関数とする．RECT 関数における，A および Q は，$N \times N$ 行列であり，それぞれ直交化の対象の行列と計算途中の値を含んだ直交化後の行列である．s は計算を開始するベクトルの位置であり $\boldsymbol{q_1}, \cdots, \boldsymbol{q_{s-1}}$ はすでに直交化されているものとする．h は計算対象のベクトルの本数であり，計算空間の四角形の高さ（height）に相当する．また，w は直交化に使用するベクトルの本数であり，計算空間の四角形の幅（width）に相当し，$\boldsymbol{q_{s-w}}$ から，$\boldsymbol{q_{s-1}}$ までのベクトルを表す．

3. 提 案 手 法

3.1　列方向分割法

2.1 節で述べた手法は，CGS 法においてベクトル $\boldsymbol{q_1}, \cdots, \boldsymbol{q_j}$ がすでに直交化されているとき，$\boldsymbol{a_{j+1}}, \boldsymbol{a_{j+2}}, \cdots, \boldsymbol{a_n}$ の直交化において，$\boldsymbol{q_1}, \cdots, \boldsymbol{q_j}$ を用いる部分を行列積で計算できるということを意味する．

文献 2) には具体的に示されていないが，CGS 法において行列積の適用範囲を拡大する手法として，容易に思いつく手法としては，図 2 における II の領域のような $(N/2) \times (N/2)$ の正方行列に対して行列積を適用するのではなく，**図 7** における II，IV，VI の領域のように，細長い長方形の行列に対して行列積を適用する手法がある．

この場合，行列ベクトル積で計算する部分は図 7 において I，III，V，VII で示される領域となり，行列積を適用できる比率を高めることができる．このアルゴリズムを以後，列方向分割法（Column-wise Blocking CGS 法，以下 CBCGS 法）と呼ぶ．

$N \times N$ 行列 A を直交化して Q を得る，CBCGS 法の疑似コードを**図 6** に示す．なお，図 6 において，M は一度に直交化するベクトルの本数であり，行列積と行列ベクトル積の性能によって決定される．s は CBCGS 法による計算を開始するベクトルの位置であり，$\boldsymbol{a_1}, \cdots, \boldsymbol{a_{s-1}}$ はそれぞれ $\boldsymbol{q_1}, \cdots, \boldsymbol{q_{s-1}}$ に直交化されているものとする．また，h は CBCGS 法を適用するベクトルの本数である．

```
begin CBCGS(A, Q, N, M, s, h)
  do i = s, h, M
    TRI(A, Q, N, i, M);
    RECT(A, Q, N, i + M, M, N − i);
  end do
end
```

図 6　列方向分割法（CBCGS 法）
Fig. 6　Column-wise blocking CGS algorithm.

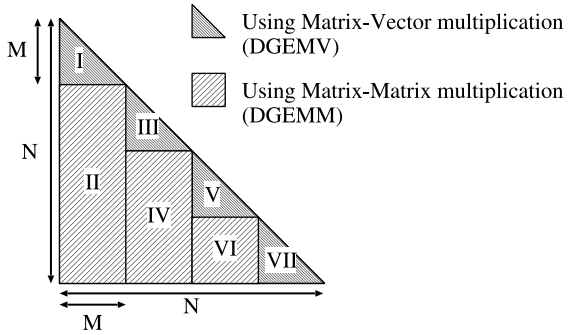

図 7　列方向分割法（CBCGS 法）における計算空間の分割方法
Fig. 7　Computational space blocking of CBCGS algorithm.

CBCGS 法により $N \times N$ 行列 A を直交化し，Q を得るには，CBCGS$(A, Q, N, M, 1, N)$ とする．

3.2　再帰分割法

図 2 で示されている，I および III の領域において，一部の計算には依存性がないことに着目することで，**図 9** における II や VI の領域のように $(N/4) \times (N/4)$ の行列積を適用することが可能になる．これを再帰的に繰り返し，行列積の大きさを適当なサイズまで小さくすることで，図 9 に示すように行列積の適用範囲を増加させることができる．このアルゴリズムを以後，再帰分割法（Recursive Blocksing CGS 法，以下 RBCGS 法）と呼ぶ．再帰分割法の疑似コードを**図 8** に示す．

図 8 において，s は RBCGS 法による計算を開始するベクトルの位置であり，$\boldsymbol{a_1}, \cdots, \boldsymbol{a_{s-1}}$

```
begin RBCGS(A, Q, N, M, s, h)
  if (h ≤ M) then
    TRI(A, Q, N, s, h);
  else
    RBCGS(A, Q, N, M, s, h/2);
    RECT(A, Q, N, s + h/2, h/2, h/2);
    RBCGS(A, Q, N, M, s + h/2, h/2);
  end if
end
```

図 8　再帰分割法（RBCGS 法）
Fig. 8　Recurisve Blocking CGS algorithm.

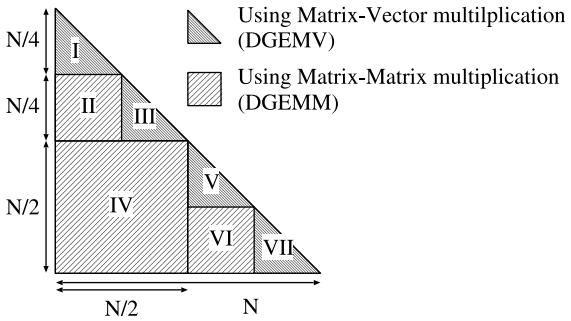

図 9　再帰分割法（RBCGS 法）における計算空間の分割方法
Fig. 9　Computational space blocking of RBCGS algorithm.

はそれぞれ $\boldsymbol{q_1}, \cdots, \boldsymbol{q_{s-1}}$ に直交化されているとする．また，h は RBCGS 法を適用するベクトルの本数である．

RBCGS 法により $N \times N$ 行列 A を直交化し，Q を得るには，RBCGS$(A, Q, N, M, 1, N)$ とする．

3.3　再帰分割法の実装

前節で示した，再帰分割法（RBCGS 法）においては，直交化対象の行列サイズを N とすると，行列積として計算する領域（RECT 関数）において，$(N/2) \times (N/2)$ の行列積を行うため，$(N/2)^2$ の作業領域が必要になるという問題点がある．この問題点を解決するために，**図 11** に示すように，大規模な行列積を行う部分を分割する．拡張した再帰分割法（Extended Recursive Block CGS 法，以下 ERBCGS 法）では，分割サイズ M の 2^k 倍の数 L を用い，$h \times h$ の行列積の領域を $(h/L)^2$ 個の $L \times L$ の行列積の領域に分割する．前節で示した RBCGS 法は，ERBCGS 法において $L = N/2$ としたときと同一である．

```
begin ERBCGS(A, Q, N, M, L, s, h)
  if (h ≤ M) then
    TRI(A, Q, N, s, h);
  else
    ERBCGS(A, Q, N, M, L, s, h/2)
    do j = 0, h/(2L), L
      do i = 0, h/(2L), L
        RECT(A, Q, N, s + iL + j, w, h)
      end do
    end do
    ERBCGS(A, Q, N, M, L, s + h/2, h/2)
  end if
end
```

図 10　拡張再帰分割法（ERBCGS 法）
Fig. 10　Extended Recursive Blocking CGS algorithm.

ERBCGS 法の疑似コードを**図 10** に示す．図 10 において，s は ERBCGS 法による計算を開始するベクトルの位置であり，$\boldsymbol{a_1}, \cdots, \boldsymbol{a_{s-1}}$ はそれぞれ $\boldsymbol{q_1}, \cdots, \boldsymbol{q_{s-1}}$ に直交化されているとする．また，h は ERBCGS 法を適用するベクトルの本数である．ERBCGS 法により，$N \times N$ 行列 A を直交化し，Q を得るには，ERBCGS$(A, Q, N, M, L, 1, N)$ とする．

CBCGS 法を大規模な行列に適用した際に，ERBCGS 法と同様に大規模な行列積演算を行う必要がある．しかし，行列サイズを N，分割サイズを M とすると，$O(NM)$ 程度の作業領域が必要になるが $M \ll N$ であるため，RBCGS 法のような分割を行う必要性が小さい．

ERBCGS 法と RBCGS 法では，計算量および通信量に関して差異がないため，本論文では以降 RBCGS 法を用いて議論を進めることとする．

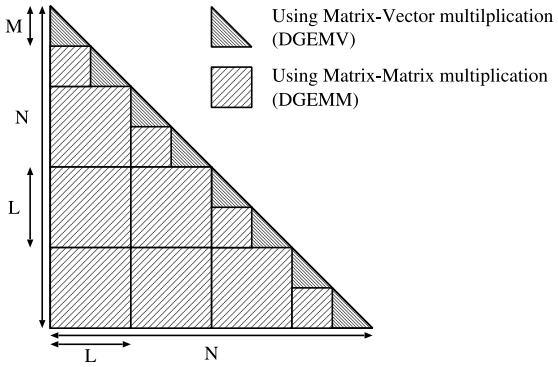

図 11　拡張再帰分割法（ERBCGS 法）における計算空間の分割方法
Fig. 11　Computational space blocking of ERBCGS algorithm.

3.4　最小分割サイズ

2.1 節で述べたように，提案手法が従来法に対して有効となるのは，行列積の性能が行列ベクトル積の性能を上回るときであるときであるといえる．図 3 に示すように，$N \times N$ の行列積において，$N = 100$ 以上では `DGEMM` を用いた行列積がつねに高性能となっている．そのため，本論文の測定環境においては，$N = 200$ 以上の行列の直交化に対して，提案手法が有効であると考えられる．

4.　並　列　化

4.1　行列の分散方法

分散メモリ型並列計算機において，行列の分散方法としていくつかの方法があるが，本論文では行方向分散（Row-wise distribution）を用いた．

行方向分散では，**図 12** に示すように，直交化対象のベクトル $\boldsymbol{a_1}, \cdots, \boldsymbol{a_N}$ および直交化済みのベクトル $\boldsymbol{q_1}, \cdots, \boldsymbol{q_N}$ の各要素がそれぞれ別々のプロセッサに分散される．そのため，ベクトルの内積の計算，各プロセッサ内で内積の部分計算を行い，その後全プロセッサでの計算結果をリダクション演算を用いることで実現できる[1)]．

他の分割方法としては，各プロセッサがベクトルの全要素を持つように分割する列方向分散（Column-wise distribution）があるが，CGS 法では内積の計算の際に，ベクトルの (本数) × (全要素) のデータのブロードキャストを必要とするため，行方向分散に比べて通信量が多くなり，CGS 法には適していないことが知られている[1)]．

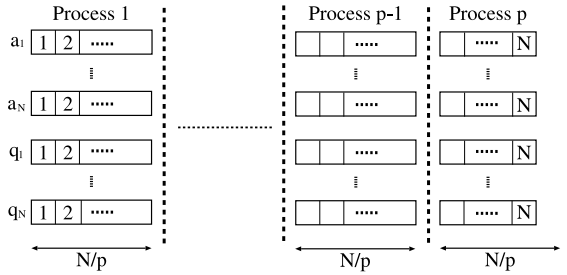

図 12　行方向分散による行列の分散
Fig. 12　Row-wise distribution.

4.2　行方向分散を用いた CGS 法

行方向分散を用いた場合，個々のプロセッサにはベクトルの一部のみが配置されるため，前述の TRI 関数，RECT 関数において，内積値の計算とノルムの計算を書き換える必要がある．行方向分散を用いた CGS 法における行列ベクトル積による計算部分（ParaTRI 関数），および行方向分散を用いた CGS 法における行列積による計算部分（ParaRECT 関数）を**図 13**，**図 14** に示す．図 13，図 14 において，$*_i^{\mathbf{local}}$ は，各プロセッサにおける値を示している．

行方向分散を用いて並列化した CBCGS 法および RBCGS 法は，それぞれのアルゴリズムにおける TRI 関数，RECT 関数を ParaTRI 関数，ParaRECT 関数にそれぞれ置き換えることで実現される．行方向分散を用いた列方向分割法（Parallel Column-wise Blocking CGS 法，以下 ParaCBCGS 法）と，行方向分散を用いた再帰分割法（Parallel Recursive Block CGS 法，以下 ParaRBCGS 法）の擬似コードをそれぞれ，**図 15**，**図 16** に示す．

5.　計算量および通信量

この章では，行方向分散を用いた列方向分割法（ParaCBCGS 法），および再帰分割法（ParaRBCGS 法）の計算量および通信量について検討する．なお，直交化は $N \times N$ 行列に対して倍精度実数で行うものとする．分割サイズを M とし，プロセッサ数を P とする．また，リダクション演算には $\log_2 P$ ステージを要すると仮定する．

5.1　計　算　量

5.1.1　行列ベクトル積による計算部分の計算量

$N \times N$ 行列に対する CGS 法において行列ベクトル積による計算部分（ParaTRI 関数）

```
begin ParaTRI(A, Q, N, s, h)
  d_s^local = Σ(q_s^local)^2
  MPI_Allreduce(d_s^local, d_s, MPI_SUM)
  q_s = q_s/√d_s
  do i = s + 1, s + h
    w^local = Q_{s,i}^{local T} a_i^local
    MPI_Allreduce(w^local, w, MPI_SUM)
    q_i^local = Q_{s,i}^local w
    d_i^local = Σ(q_i^local)^2
    MPI_Allreduce(d_i^local, d_i, MPI_SUM)
    q_i = q_i/√d_i
  end do
end
```

図 13　行方向分散を用いた CGS 法における行列ベクトル積による計算部分（ParaTRI 関数）
Fig. 13　Row-wise distribution CGS's part of matrix vector multiplication (ParaTRI function).

```
begin ParaRECT(A, Q, N, s, w, h)
  S^local = Q_{s+w,s+h}^{T^local} A_{s+w,s+h}
  MPI_Allreduce(S^local, S, MPI_SUM)
  Q_{s+w,s+h}^local = S Q_{s+w,s+h}^local
end
```

図 14　行方向分散を用いた CGS 法における行列積による計算部分（ParaRECT 関数）
Fig. 14　Row-wise distribution CGS's part of matrix multiplication (ParaRECT function).

では，ノルム演算（NRM2(N)）および定数倍（SCAL(N)）が h 回行われる．NRM2(N) の演算量は $2N$，SCAL(N) の演算量は N であるため，TRI 関数全体での 2 つのレベル 1 BLAS ルーチンの演算量は $3Nh$ と表すことができる．

また，行列ベクトル積 $\boldsymbol{y_1} = A_1\boldsymbol{x_1}$（$\boldsymbol{y_1} \in R^i$，$A_1 \in R^{i\times N}$，$\boldsymbol{x_1} \in R^j; 2 \le i \le h$）および $\boldsymbol{y_2} = A_2\boldsymbol{x_2}$（$\boldsymbol{y_2} \in R^N$，$A_2 \in R^{N\times i}$，$\boldsymbol{x_2} \in R^i; 2 \le i \le h$）がそれぞれ $h-1$ 回行われる．2 回の行列ベクトル積の演算量はどちらも $2N \cdot i$ であるため，TRI 関数全体での行列ベクトル積の部分の演算量は $\sum_{i=2}^{h-1} 4N \cdot j = 4N\{h(h-1)/2 - 1\}$ と表すことができる．したがって，TRI 関数の全体の計算量を $\mathrm{TRI}_{\mathrm{comp}}(N,h)$ とすると，

$$\begin{aligned}\mathrm{TRI}_{\mathrm{comp}}(N,h) &= 3Nh + 4N\{h(h-1)/2 - 1\} \\ &= N(2h^2 + h - 4)\end{aligned} \tag{11}$$

となる．

```
begin ParaCBCGS(A, Q, N, M, s, h)
  do i = s, h, M
    ParaTRI(A, Q, N, i, M)
    ParaRECT(A, Q, N, i + M, M, N − i)
  end do
end
```

図 15　行方向分散を用いた列方向分割法（ParaCBCGS 法）
Fig. 15　Row-wise distribution Parallelized Column-wise Blocking CGS (ParaCBCGS) algorithm.

```
begin ParaRBCGS(A, Q, N, M, s, h)
  if (h ≤ M) then
    ParaTRI(A, Q, N, s, h)
  else
    ParaRBCGS(A, Q, N, M, s, h/2)
    ParaRECT(A, Q, N, s + h/2, h/2, h/2)
    ParaRBCGS(A, Q, N, M, s + h/2, h/2)
  end if
end
```

図 16　行方向分散を用いた再帰分割法（ParaRBCGS 法）
Fig. 16　Row-wise distribution Parallelized Recursive Blocking CGS (ParaRBCGS) algorithm.

5.1.2　行列積による計算部分の計算量

$N \times N$ 行列に対する CGS 法において行列積による計算部分（ParaRECT 関数）では，$C_1 = A_1B_1$（$C_1 \in R^{h\times w}$，$A_1 \in R^{h\times N}$，$B_1 \in R^{N\times w}$）および $C_2 = A_2B_2$（$C_2 \in R^{N\times h}$，$A_2 \in R^{N\times w}$，$B_2 \in R^{w\times h}$）の 2 つの行列積が行われる．2 回の行列積（$C_1 = A_1B_1$，$C_2 = A_2B_2$）の演算量は，それぞれ $2Nhw$ であるため，RECT 関数の全体の計算量を $\mathrm{RECT}_{\mathrm{comp}}(N,w,h)$ とすると，

$$\mathrm{RECT}_{\mathrm{comp}}(N, w, h) = 4Nhw \tag{12}$$

となる．

5.1.3　列方向分割法の計算量

列方向分割法（ParaCBCGS 法）では，ParaTRI 関数が (N/M) 回と ParaRECT 関数が $(N/M)-1$ 回呼ばれる．そのため，分割サイズを M としたとき $N \times N$ 行列に対する CBCGS 法の計算量を $\mathrm{CBCGS}_{\mathrm{comp}}(N, M)$ とすると，

$$\begin{aligned}&\mathrm{CBCGS}_{\mathrm{comp}}(N, M)\\ &= (N/M)\cdot \mathrm{TRI}_{\mathrm{comp}}(N, M) + \sum_{i=1}^{(N/M)-1} \mathrm{RECT}_{\mathrm{comp}}(N, M, N-iM)\\ &= 2N^3 + (1-4/M)N^2\end{aligned} \tag{13}$$

となる．

5.1.4　再帰分割法の計算量

再帰分割法（ParaRBCGS 法）は図 8 に示したように再帰的に定義されるため，分割サイズを M としたとき $N \times N$ 行列に対する ParaRBCGS 法の計算量を $\mathrm{RBCGS}_{\mathrm{comp}}(N, M)$ とすると，$N = 2^k M$ のとき，

$$\begin{aligned}&\mathrm{RBCGS}_{\mathrm{comp}}(N, M)\\ &= (N/M)\cdot \mathrm{TRI}_{\mathrm{comp}}(N, M) + \sum_{i=1}^{k} 2^{k-i}\mathrm{RECT}_{\mathrm{comp}}(N, 2^{i-1}M, 2^{i-1}M)\\ &= 2N^3 + (1-4/M)N^2\end{aligned} \tag{14}$$

となる．

5.2　通　信　量

5.2.1　行列ベクトル積による計算部分の通信量

$N \times N$ 行列に対する CGS 法における，h 本のベクトルに対する行列ベクトル積による計算部分（ParaTRI 関数）の通信は，ベクトルのノルムの計算に 1 回あたり 8 バイトの MPI_Allreduce が h 回行われ，2 回の行列ベクトル積にはさまれた部分に 1 回あたり $8 \cdot i$ $(i = 2, \cdots, h)$ バイトの MPI_Allreduce が $h-1$ 回行われる．プロセッサ数を P とすれば，リダクション演算には $\log P$ 回の通信が必要であるため，ParaTRI 関数の通信量を $\mathrm{TRI}_{\mathrm{comm}}(P, h)$ とすると，

$$\begin{aligned}\mathrm{TRI}_{\mathrm{comm}}(P, h) &= (8h + \sum_{i=2}^{h-1} 8\cdot i)\log P\\ &= 4(h^2 + 3h - 2)\log P\end{aligned} \tag{15}$$

となる．

5.2.2　行列積による計算部分の通信量

$N \times N$ 行列に対する CGS 法における，直交化済みの w 本のベクトルを用いた h 本のベクトルに対する行列積による計算部分（ParaRECT 関数）では，$w \cdot h$ 個のデータに対して 1 回の MPI_Allreduce があるので通信量は，$8hw$ となる．プロセッサ数を P とすれば，ParaRECT 関数の通信量，$\mathrm{RECT}_{\mathrm{comm}}(P, w, h)$ は

$$\mathrm{RECT}_{\mathrm{comm}}(P, w, h) = 8hw \cdot \log P \tag{16}$$

となる．

5.2.3　列方向分割法の通信量

$N \times N$ 行列に対する列方向分割法（ParaCBCGS 法）では，分割サイズを M とすると，M 本のベクトルに対する ParaTRI 関数が N/M 回，高さが M ずつ減少する ParaRECT 関数が $(N/M)-1$ 回呼ばれる．そのため，プロセッサ数 P での ParaCBCGS 法の通信量を $\mathrm{CBCGS}_{\mathrm{comm}}(P, N, M)$ とすると，

$$\begin{aligned}&\mathrm{CBCGS}_{\mathrm{comm}}(P, N, M)\\ &= (N/M)\mathrm{TRI}_{\mathrm{comm}}(P, M) + \sum_{i=1}^{(N/M)-1} \mathrm{RECT}_{\mathrm{comm}}(P, M, N-iM)\\ &= 4N(N + 3 - 2/M)\log P\end{aligned} \tag{17}$$

となる．

5.2.4　再帰分割法の通信量

$N \times N$ 行列に対する再帰分割法（ParaRBCGS 法）では，計算量と同様に再帰的に計算され，$N = 2^k M$ のとき，プロセッサ数 P での ParaRBCGS 法の通信量を $\mathrm{RBCGS}_{\mathrm{comm}}(P, N, M)$ とすると，

$$\begin{aligned}&\mathrm{RBCGS}_{\mathrm{comm}}(P, M)\\ &= (N/M)\mathrm{TRI}_{\mathrm{comm}}(P, M) + \sum_{i=1}^{k} 2^{k-i}\mathrm{RECT}_{\mathrm{comm}}(P, 2^{i-1}M, 2^{i-1}M)\end{aligned}$$

$$= 4N(N + 3 - 2/M) \log P \tag{18}$$

となる．

5.3 議　　論

計算量の点では，5.1 節で示したように，列方向分割法（CBCGS 法）と再帰分割法（RBCGS 法）には差がない．しかし，行列積の計算において図 9 の II の領域のような正方行列に対する計算の方が，図 7 の II の領域のような矩形の行列に対する計算より，メモリとキャッシュとのデータ転送の点で有利[7] であるため，単一プロセッサにおいては，RBCGS 法の方が高速になると考えられる．

一方，通信量の点においては，5.2 節において，式 (17) および，式 (18) で示したように，ParaCBCGS 法と ParaRBCGS 法の通信量に差がないため，単一プロセッサと同様に ParaRBCGS 法の方が高速になると考えられる．

しかし，より大規模な行列においては図 11 に示すような，拡張再帰分割法（ERBCGS 法）を用いるため，通信回数が ParaCBCGS 法に比べ，$(N/L)^2$ に比例して増加し，通信レイテンシによるオーバヘッドにより，ParaCBCGS 法および ParaRBCGS 法の性能差は小さくなると予想される．

6. 性 能 評 価

行方向分散を用いた CGS 法の性能評価には 32 ノードの PC クラスタを用い，プロセッサ数を固定して行列サイズを変化させたときと，行列サイズを固定してプロセッサ数を変化させたときについて，レベル 2 BLAS である行列ベクトル積（`GEMV`）を用いた naive な実装，列方向分割法（ParaCBCGS）法，そして再帰分割法（ParaRBCGS）法の性能を比較した．

直交化対象の行列の各要素は $[0, 1]$ の一様乱数を用い，直交化の計算精度の評価指標としては $||Q^T Q - I||_F$ を用いた．$||*||_F$ はフロベニウスノルムであり，I は単位行列である．評価環境を**表 1** に示す．プログラムは C で記述し，コンパイラは Intel C Compiler（`icc`, Version 10）を用い，最適化オプションとして “`-O3 -xP`” を用いた．クラスタの各ノードは，1000Base-T の Gigabit Ethernet で接続されており，2 ノード間の通信において最大バンド幅は 112.2 MB/s であり，レイテンシは 24 マイクロ秒程度である．

6.1 プロセッサ数に対する評価

プロセッサ数を $P = 2$，4，8，16，32 と変化させ，その経過時間を測定した．行列サイズ $N = 8000$ のときの評価結果を **図 17** に示す．分割サイズは $M = 64$ とした．RBCGS 法の実装には ERBCGS 法を用い，最大分割サイズは $L = 2048$ とした．CBCGS 法および RBCGS 法では，プロセッサ数の増加にともなって性能向上が頭打ちになっている．これは，データを行方向分散したため，プロセッサ数の増加にともなって，個々のプロセッサにおける行列のサイズが減少し計算時間が短縮される一方で，通信量が $\log P$ に比例して増加するためである．したがって，プロセッサ数の増加に対し行列サイズを増加させることで，各 CGS 法の性能を向上させることができると考えられる．

表 1　評価環境
Table 1　Evaluation environment.

Platform	Xeon PC cluster
Number of Nodes	32
CPU Type	Xeon 3 GHz
L1 Cache	I-Cache: 12 K uops D-Cache: 16 KB
L2 Cache	2 MB
Main Memory	DDR-SDRAM 1 GB
OS	Linux 2.6.20-1.2933.fc6smp
MPI	OpenMPI 1.2.3
BLAS	GotoBLAS r1.19

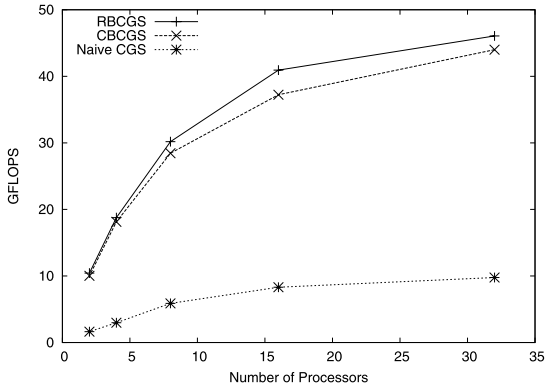

図 17　行列サイズ $N = 8000$ における各 CGS 法の評価結果
Fig. 17　Performance results of CGS algorithms (Number of processes vs. GFLOPS; Matrix size $N = 8000$).

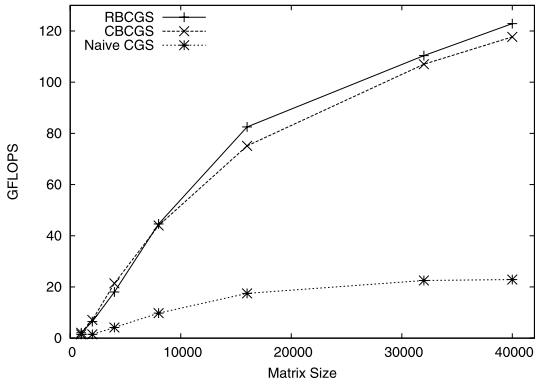

図 18　プロセッサ数 $P = 32$ における各 CGS 法の評価結果
Fig. 18　Performance results of CGS algorithms (Matrix size vs. GFLOPS; Number of processes $P = 32$).

6.2　行列サイズに対する評価

次に，行列サイズを $N = 1000$ から $N = 40000$ まで変化させ，その経過時間を測定した．プロセッサ数 $P = 32$ のときの各 CGS 法の評価結果を**図 18** に示す．ブロックサイズは $M = 64$ とした．RBCGS 法の実装には ERBCGS 法を用い，最大分割サイズは $L = 2048$ とした．

図 18 から分かるように，行列サイズ $N = 10000$ 以下の領域においては，CBCGS 法が RBCGS 法に対して高性能となっている．これは，5.3 節において議論したように，小さい行列サイズにおいては RBCGS 法の通信回数が CBCGS 法に対して多いからである．

図 19 に提案手法である再帰分割法（RBCGS 法）において，プロセッサ数を増加させたときの各行列サイズに対する関係を示す．

図 19 において，行列サイズ $N = 20000$ のときプロセッサ数 $P = 32$ において，プロセッサ数 $P = 16$ に対して，約 1.4 倍の性能向上が得られており，提案手法は大規模な行列に対して，プロセッサ数を増加させることで性能改善ができることを示している．

6.3　精 度 評 価

図 20 は，各 CGS 法において，行列サイズを $N = 1000$ から $N = 40000$ まで変化させたときの，直交化の計算精度の評価指標とした $||Q^TQ - I||_F$ の値を示したものである．

RBCGS 法，および CBCGS 法は naive な CGS 法に対し，計算精度の点で著しい変化はなく，計算順序の変更による影響は見られない．

図 18 から，naive な手法に対し RBCGS 法が 8000 以上の行列サイズにおいて最も高速であることが分かる．特に，プロセッサ数 $P = 32$，行列サイズ $N = 40000$ の場合に RBCGS 法は naive な実装に比べると約 5.36 倍高速となった．また，RBCGS 法は，CBCGS 法に対して，約 4.4%高速な結果が得られた．

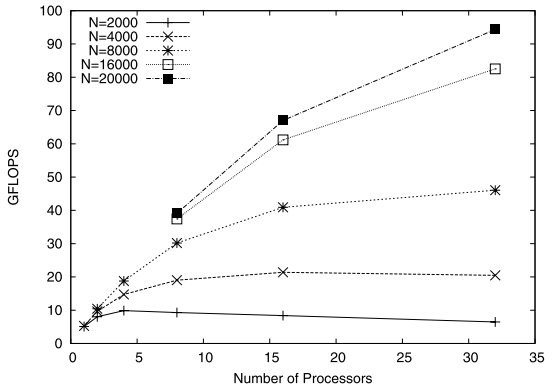

図 19　各行列サイズにおけるプロセッサ数を増加させたときの RBCGS 法の評価結果
Fig. 19　Performance results of RBCGS algorithm.

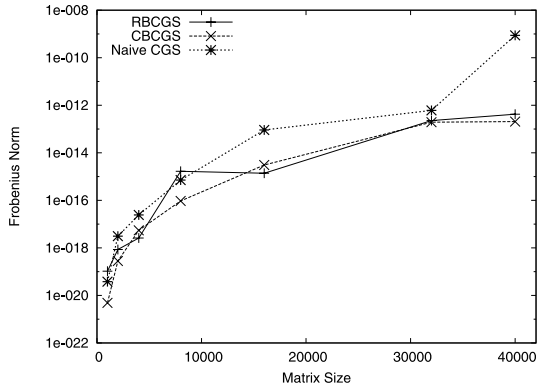

図 20　プロセッサ数 $P = 32$ における直交化精度
Fig. 20　Loss of orthogonality of CGS algorithms (Number of processes $P = 32$).

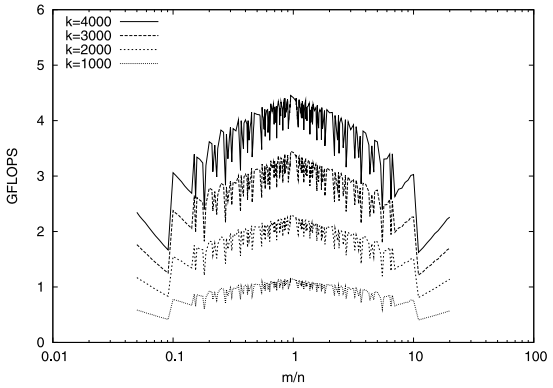

図 21　DGEMM における行列のサイズの比率による性能比較
Fig. 21　Performance comparison of DGEMM's row-column ratio.

CBCGS 法および，RBCGS 法では RECT 関数において，2 回の行列積 $C_1 = A_1B_1$（$A_1 \in R^{m\times k}$, $B_1 \in R^{n\times k}$, $C_1 \in R^{m\times n}$），および $C_2 = A_2B_2$（$A_2 \in R^{m\times n}$, $B_2 \in R^{k\times m}$, $C_2 \in R^{n\times k}$）という計算が行われる．このとき，$1 \le l \le N/M$ とすると，RBCGS 法では，$(m,n,k) = (lM, lM, N)$ となるのに対して，CBCGS 法では，$(m,n,k) = (M, lM, N)$ となる．この (m,n,k) の組合せによって，性能に差が現れると考えられる．

6.4　行列積（DGEMM）の行列サイズに対する評価

前節で検討した CGS 法の性能差について検証するため，行列積（DGEMM）の行列サイズに対する性能について評価を行った．**図 21** に評価結果を示す．図 21 のグラフは DGEMM（$C = \alpha A^T B + \beta C$;　$\{A \in R^{k\times m}, B \in R^{k\times n}, C \in R^{m\times n}\}$）の呼び出しにおいて $\alpha = 1.0$, $\beta = 0.0$ として，k の値を 1000 から 4000 まで変化させ，m，n の値をそれぞれ，50 から 4000 まで 50 刻みで変化させたときの結果を，横軸に m/n の値，縦軸にすべての m/n に対しての処理性能の最高値をとったものである．k の値によらず，$m/n = 1$ において最も高速になっており，m，n の比が減少もしくは増加する方向に対して，処理性能が低下する傾向が見られる．そのため，DGEMM の実行回数のうち，$l = 1$ の一度だけ $m/n = 1$ となる CBCGS 法に対して，すべての l において $m/n = 1$ となる RBCGS 法の方が処理性能が高くなると考えられる．

7. ま と め

本論文では，直交化アルゴリズムの 1 つである古典 Gram-Schmidt 法（CGS 法）の PC クラスタ環境における効率的な実装方法について提案した．

CGS 法においては，内積計算とベクトル変換を行列積に変更することで高速化できることが知られているが，この手法を拡張し行列積を適用する範囲を増やすことで性能を改善できることを示した．

また，行列積（DGEMM）において，正方行列を用いる場合が高性能であることを明らかにし，行列積を用いる性能改善においては，正方行列を用いるように分割を行う再帰分割法が，高性能となることを示した．

提案手法を行方向分散を用いて並列化するとともに，PC クラスタに実装し，性能評価を行った結果，32 ノードの Xeon 3 GHz PC クラスタでは行列サイズ $N = 40000$ において naive な実装に対し約 5.36 倍高速化され，約 122.9 GFLOPS の性能を得ることができた．特に，プロセッサ数 $P = 32$，行列サイズ $N = 40000$ において，正方行列に対する行列積を用いる再帰分割（RBCGS）法は，細長い行列に対する行列積を用いる列方向分割法（CBCGS）法に対して，約 4.4%高速な結果が得られた．

参 考 文 献

1) Katagiri, T.: Performance Evaluation of Parallel Gram-Schmidt Re-orthogonalization Methods, *Proc. 5th International Meeting on High Performance Computing for Computational Science* (*VECPAR 2002*), Lecture Notes in Computer Science, No.2565, pp.302–314, Springer-Verlag (2003).
2) 寒川　光：RISC 超高速化プログラミング技法，共立出版 (1995).
3) Yokozawa, T., Takahashi, D., Boku, T. and Sato., M.: Efficient Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication, *Proc. 4th International Workshop on Parallel Matrix Algorithms and Applications* (*PMAA'06*), pp.37–38 (2006).
4) Daniel, J., Gragg, W.B., Kaufman, L. and Stewart, G.W.: Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization, *Math. Comput.*, Vol.30, pp.772–795 (1976).
5) Lingen, F.J.: Efficient Gram-Schmidt orthonormalisation on parallel computers, *Communications in Numerical Methods in Engineering*, Vol.16, pp.57–66 (2000).
6) 直野　健，猪貝光祥，木立啓之：数値計算ポリシー入力型グラムシュミット直交化ライブラリの異種混合計算機環境における性能評価，情報処理学会論文誌：コンピュー

ティングシステム，Vol.46, No.SIG 12(ACS11), pp.279–287 (2005).
7) Goto, K. and van de Geijn, R.: Anatomy of High-Performance Matrix Multiplication, *ACM Trans. Math. Softw.*, Vol.34 (2008).

(平成 19 年 10 月 9 日受付)
(平成 20 年 1 月 21 日採録)

横澤　拓弥

昭和 58 年生．平成 16 年群馬工業高等専門学校電子情報工学科卒業．平成 18 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科博士前期課程在学中．

高橋　大介（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．平成 16 年同大学大学院システム情報工学研究科講師．平成 18 年同助教授，平成 19 年同准教授．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度，平成 15 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．

朴　　泰祐（正会員）

昭和 35 年生．昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，平成 16 年同大学大学院システム情報工学系助教授，平成 17 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，グリッドに関する研究に従事．平成 14 年度および平成 15 年度情報処理学会論文賞受賞．日本応用数理学会，IEEE CS 各会員．

佐藤　三久（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学システム情報工学研究科教授．平成 19 年より，同大学計算科学研究センター長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．IEEE，日本応用数理学会各会員．