大規模密行列に対する古典Gram-Schmidt直交化の高速化

横 澤 拓 弥

筑波大学大学院システム情報工学研究科

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一つである。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，固有値問題の計算などにおいて広く用いられている手法である。

Gram-Schmidt の直交化を用いて行列の対角化を行う場合 $N \times N$ 列の行列に対して $O(N^3)$ の計算量を必要とするため，Gram-Schmidt の直交化を高速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 [2]。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する CGS 法では，内積計算およびベクトル変換を行列積に変換することでさらに高速化する手法が提案され，すでにいくつかの手法について評価がなされている [3]。

MGS 法を用いることで CGS 法における精度の問題が解決できるが，MGS 法は計算順序の依存性により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）[1] を用いることで，CGS 法においても精度を改善できることが知られている。

本稿では，大規模行列に対する CGS 法の高速化手法を提案し，HITACHI SR11000 において評価した結果について述べる。

2 古典 Gram-Schmidt 直交化法

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。

```
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

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。

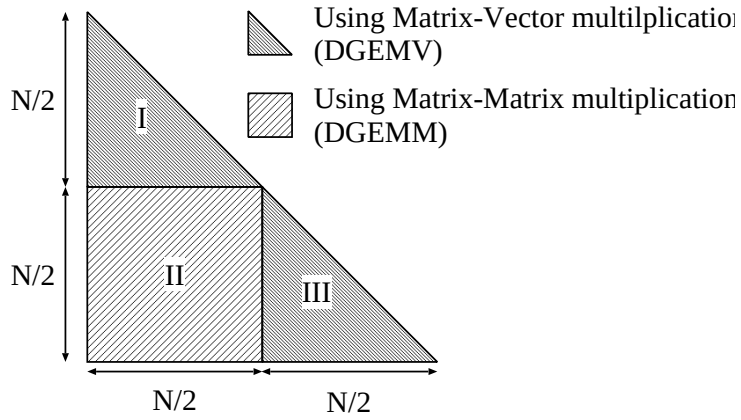

図 2: $N \times N$ 行列に対する行列積を用いた CGS 法の計算空間

2.1 行列積を用いた古典 Gram-Schmidt 直交化法

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 [4]。

この手法を $N \times N$ 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共に，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 である行列積で計算されるが，I および III で示されている範囲はレベル 2 BLAS である行列ベクトル積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。そのため，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により，演算性能を向上させることが可能であると考えられる。

本節では 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示したが，以下ではそれぞれの部分について説明する。

2.2 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 3 のように書くことができる。

以降，行列ベクトル積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）に相当し，$Q_{i,j}$ 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。

```
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 3: 行列ベクトル積を用いる計算部分（TRI 関数）

```
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 4: 行列積を用いる計算部分 （RECT 関数）

2.2.1 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 4 のように書くことができる。

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に相当し，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に相当する。

3 CGS 法の分割

3.1 列方向分割法

2 章では，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，直交化されたベクトルを用いる部分を行列積による計算に変換できることを述べた。

CGS 法において行列積の適用範囲を拡大する手法として，図 5 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 5 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。

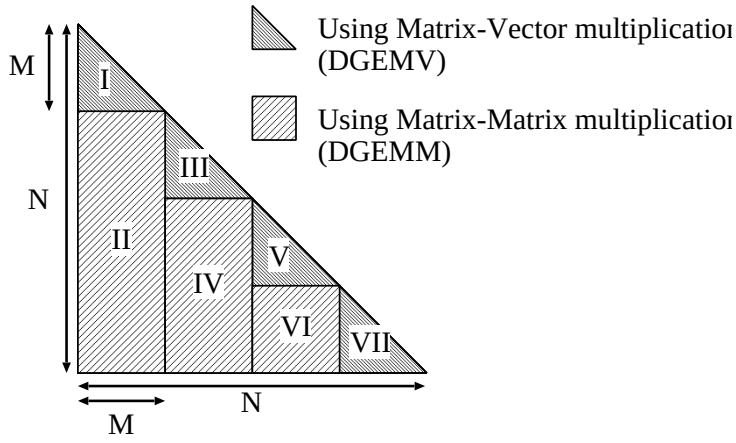

図 5: 列方向分割法（CBCGS 法）における計算空間の分割方法

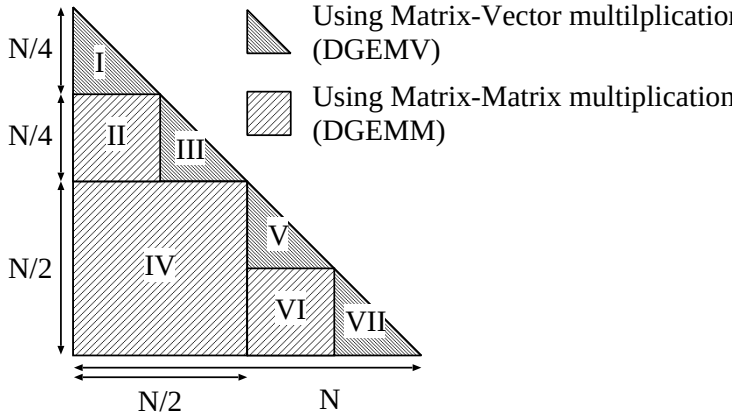

図 6: 再帰分割法（RBCGS 法）における計算空間の分割方法

3.2 再帰分割法

大規模な行列に対して前節で示した列方向分割法を適用した場合，行列サイズ N に対して行列ベクトル積を用いる部分のサイズ M が $N \gg M$ となるため，行列積を用いる計算部分における行列の列と行の比が大きくなり，行列積の計算性能が低下する。

そこで，直交化の対象の行列のうち半数を行列ベクトル積を用いて計算し，残り半数を行列積と行列ベクトル積を用いて計算するようにすることで，行列積を用いた計算部分の行列を正方行列とすることができる。

このとき，行列ベクトル積を用いた計算部分において，一部の計算には依存性がないことに着目することで，図 6 における II や VI の領域のように再び行列積による計算が可能になる。

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 6 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。

```
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 7: 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）

```
begin ParaRECT(A, Q, N, s, w, h)
  S^local = Q_{s+w,s+h}^{local^T} 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 8: 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）

4 並列化における行列の分散方法

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，本稿では行方向分散（Row-wise distribution）を用いた。このとき行方向分散を用いることにより，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ため，CGS 法におけるベクトルの内積計算が，各プロセッサ内で部分計算を行い，その後全プロセッサでの計算結果に対してリダクション演算を用いることで実現できる 。

行列積を用いた CGS 法においては，2 章において示した行列ベクトル積を用いる計算部分（TRI 関数）および行列積を用いた計算部分（RECT 関数）をそれぞれ図 7，図 8 に示すように拡張することで実現される。

5 性能評価

行方向分散を用いた CGS 法の性能評価には HITACHI SR11000 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（CBCGS）法，そして再帰分割法（RBCGS）法の性能を比較した。

プログラムは MPI を用いて並列化を行い，C 言語で実装した。また，BLAS ライブラリには

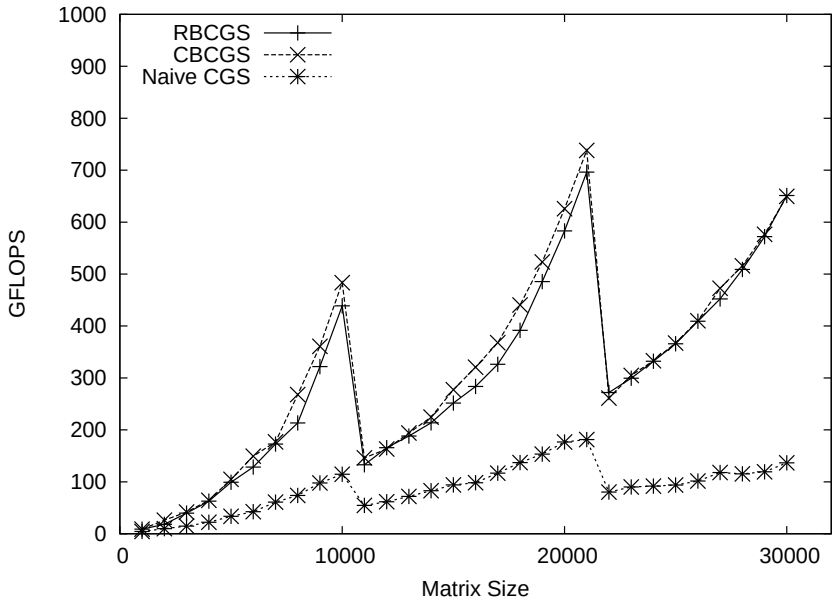

図 9: プロセッサ数 256（ノード数 16）における行列サイズの増加に対する評価結果

ノード内並列化を行わないスカラー版を用い，コンパイル時の最適化オプションとして `-Oss` を用いた。

5.1 行列サイズの増加に対する評価

5.1.1 プロセッサ数 256 における評価

図 9 にプロセッサ数 256（ノード数 16）における行列サイズを増加させたときの評価結果を示す。

どのアルゴリズムにおいても行列サイズの増加に伴って，性能が向上する傾向が見られた。

行列積を用いた CBCGS 法及び RBCGS 法はいずれのサイズにおいても行列ベクトル積を用いた naive な手法に対して性能が向上している。これは，レベル 3 BLAS である行列積を用いたことによる性能向上であると考えられる。

行列サイズ 21000 以下では，CBCGS 法が RBCGS 法に対し約 5%高速な結果が得られた。一方で行列サイズ 22000 以上においては，CBCGS 法と RBCGS 法の差はおよそ 5%以内となった。

5.1.2 プロセッサ数 1024 における評価

図 10 に SR11000 における最大プロセッサ数である，プロセッサ数 1024（ノード数 64）における行列サイズを増加させたときの評価結果を示す。

プロセッサ数 256 における結果と同様に，いずれのアルゴリズムにおいても行列サイズの増加に伴って，性能が向上する傾向が見られた。特に，行列サイズ 40000 において，CBCGS 法および RBCGS 法は，それぞれ約 1.04TFLOPS，約 1.22TFLOPS の結果が得られ，naive な手法に対しそれぞれ，約 5.3 倍，約 6.2 倍の性能向上となった。どの行列サイズにおいても RBCGS 法が

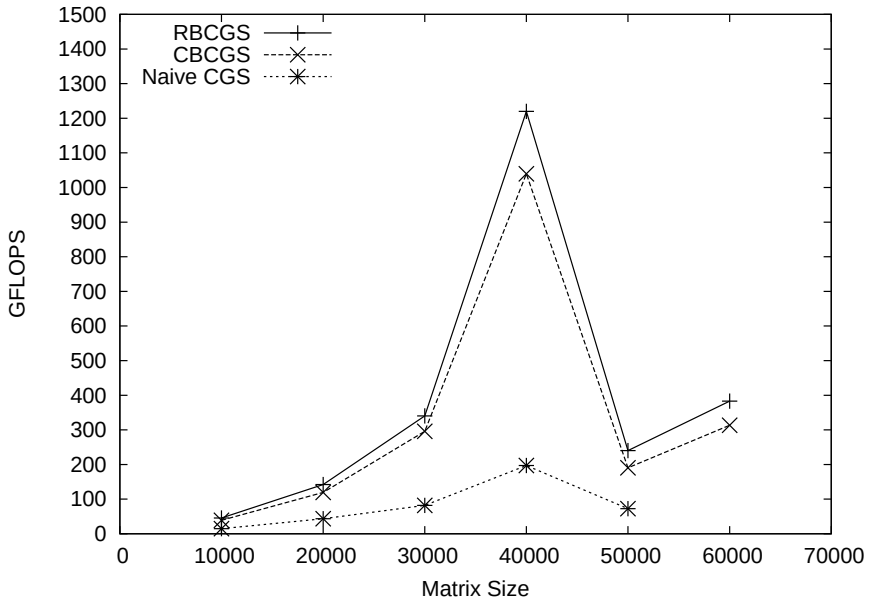

図 10: プロセッサ数 1024（ノード数 64）における行列サイズの増加に対する評価結果

CBCGS 法に対し，およそ 15%以上高速な結果が得られた。特に，行列サイズ 50000 において 約 26%高速な結果が得られた。これは，プロセッサ数 512 における結果と同様に CBCGS 法において，行列積を用いた計算部分の行列の列と行の比が大きくなったことによる性能低下の影響と考えられる。プロセッサ数 512 における結果と比べて，RBCGS 法の性能向上幅が大きくなった理由としては，プロセッサ数の増加により，それぞれのプロセッサにおける行列サイズが小さくなったためと考えられる。

5.2 ノード数の増加に対する評価

図 11 に行列サイズ 30000 におけるノード数（プロセッサ数）を増加させたときの評価結果を示す。

行列サイズ 30000 では，いずれのアルゴリズムにおいてもノード数 16（プロセッサ数 256）において最も高速な結果が得られている。しかし，ノード数 32 以上においては，性能が低下している。これは，プロセッサ数の増加による通信時間の増大，およびそれぞれのプロセッサにおける行列サイズの縮小による行列積の性能低下によるものと考えられる。

図 10 に示したように，プロセッサ数 1024，行列サイズ 40000 に対して RBCGS 法を用いたとき約 1.22TFLOPS の性能が得られることから，より大規模な行列に対しては，プロセッサ数を増加させることにより，高い処理性能を得られると考えられる。

6 まとめ

本稿では，大規模行列に対する CGS 法の高速化手法を提案し，SR11000 において評価した結果について述べた。

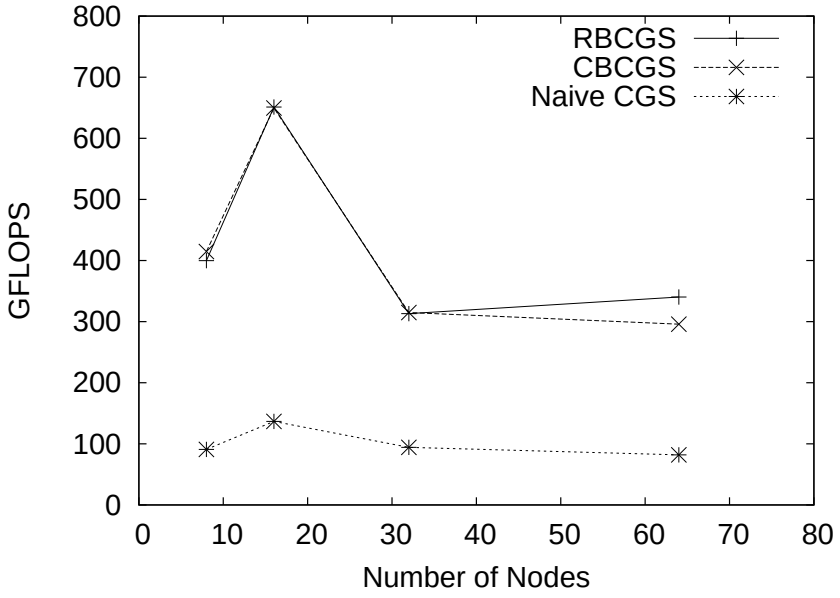

図 11: 行列サイズ 30000 におけるノード数（プロセッサ数）を増加させたときの評価結果

行列積を用いた CGS 法において，列と行の比の大きい行列を用いる列方向分割法と，正方行列を用いる再起分割法とを行方向分散を用いて並列化行い，その評価を行った。

その結果，プロセッサ数 1024，行列サイズ 40000 において，正方行列を用いるように分割を行う再起分割法を用いることで，約 1.22TFLOPS の結果が得られ，行列積を用いない手法に対して，約 6.2 倍の性能向上が得られた。また，列方向分割法に対しては，約 17%高速な結果となった。

参考文献

[1] J.W.Daniel, W. B. Gragg, L.Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Mathematics of Computation*, 30(136):772–795, Oct 1976.

[2] Takahiro Katagiri. Performance evaluation of parallel Gram-Schmidt re-orthogonalization methods. In José M. L. M. Palma, Jack Dongarra, Vicente Hernández, and A. Augusto de Sousa, editors, *VECPAR*, volume 2565 of *Lecture Notes in Computer Science*, pages 302–314. Springer, 2002.

[3] Takuya Yokozawa, Daisuke Takahashi, Taisuke Boku, and Mitsuhisa Sato. Efficient parallel implementation of classical Gram-Schmidt orthogonalization using matrix multiplication. In *Proc. 4th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'06)*, pages 37–38, 2006.

[4] 寒川光. *RISC* **超高速化プログラミング技法**. 共立出版, 1995.