社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－106 (6)
2006／6／12

行列積を用いた古典 Gram-Schmidt 直交化の並列化手法の検討

横澤 拓弥† 高橋 大介†
朴 泰祐† 佐藤 三久†

本論文では，直交化アルゴリズムの一つである古典 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を増やすと共に，再帰的に行列積を行うことで，性能を改善することができることを示す．提案する手法を dual Xeon SMP クラスタに実装し，性能評価を行った．その結果，8 ノードの dual Xeon 2.4 GHz PC SMP クラスタでは約 7.89 GFLOPS の性能を得ることができた．

Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication

Takuya Yokozawa,† Daisuke Takahashi,† Taisuke Boku† and Mitsuhisa Sato†

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 a recursive matrix multiplication improves performance effectively. We succeeded in obtaining performance of approximately 7.89 GFLOPS on an 8-node dual Xeon 2.4 GHz PC SMP 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一つである．Gram-Schmidt の直交化は，直交化処理を行う手法として知られており，固有値問題や連立一次方程式の反復法におけるクリロフ部分空間の計算などにおいて広く用いられているアルゴリズムである．

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 $n \times n$ の行列に対して $O(n^3)$ 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)~4)]．また，Gram-Schmidt の直交化の並列アルゴリズムも多く提案されている[5)~8)]．

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），そして，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（以下 DGKS 法）[2)] が知られている．

本論文では，高速に計算が行えると共に，並列化にも向いている CGS 法について取り扱うことにする．なお，精度については DGKS 法を用いることで改善することが可能である．

CGS 法においては，内積計算とベクトル変換を行う必要があるが，これらの計算は BLAS 2 アルゴリズムである行列ベクトル積を用いることができ，ある程度はキャッシュの再利用を図ることが可能である．

一方，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 BLAS 3 アルゴリズム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[4)]．

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を増やすと共に，行列積を再帰的に適用することで，性能を改善することができることを示す．

以下，2 章で Gram-Schmidt 直交化アルゴリズムに

† 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

```
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 (CGS) 法

```
do j = 1,n
  a_j^(0) = a_j
  do i = 1,j-1
    a_j^(i) = a_j^(i-1) - (q_i, a_j^(i-1))q_i
  end do
  q_j = a_j^(j-1); q_j = q_j/||q_j||
end do
```

図 2 修正 Gram-Schmidt (MGS) 法

ついて説明する．3 章で CGS 法を行列積で行う手法の拡張について述べる．4 章で CGS 法の並列化について述べる．5 章で性能評価の結果を示す．最後の 6 章はまとめである．

2. Gram-Schmidt 直交化アルゴリズム

Gram-Schmidt の直交化として知られている，CGS 法と MGS 法について，以下に説明する．

なお，(q_i, a_j) は，ベクトル q_i と a_j の内積を，$\|q_j\|$ は，ベクトル q_j のユークリッドノルムを表している．

2.1 古典 Gram-Schmidt (CGS) 法

m 行 n 列の行列 $A = (a_1 a_2 \cdots a_n)$ に対する CGS 法は，図 1 のように記述される．

図 1 で示した CGS 法においては，q_j を計算する際に BLAS 2 アルゴリズムである行列ベクトル積 (GEMV) を用いることができ，ある程度はキャッシュの再利用を図ることが可能である．

2.2 修正 Gram-Schmidt (MGS) 法

MGS 法は，CGS 法における計算誤差の蓄積を抑えて精度を向上させるアルゴリズムとして知られている．m 行 n 列の行列 $A = (a_1 a_2 \cdots a_n)$ に対する MGS 法は，図 2 のように記述される．

図 2 で示した MGS 法においては，$a_j^{(i)}$ と $a_j^{(i-1)}$ の間に依存性があるので，内積計算を BLAS 1 アルゴリズムである DOT ルーチンで，ベクトル変換を同じく BLAS 1 アルゴリズムである AXPY ルーチンでしか計算することができない．

BLAS 1 アルゴリズムは，キャッシュの再利用を図ることが困難であり，計算速度の観点からは CGS 法に比べて不利となる．

3. 古典 Gram-Schmidt (CGS) 法を行列積で行う手法の拡張

3.1 従来の手法

図 1 で示した CGS 法において，複数のベクトル $q_1, q_2, \cdots, q_{j-1}$ の直交化が既に済んでいれば，$a_j, a_{j+1}, \cdots, a_n$ との内積計算には依存性がないため，$q_1, q_2, \cdots, q_n$ 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]．

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

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

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

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

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

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

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

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

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

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

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

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

ここで，$Q_{13} = (q_1 q_2 q_3)$, $Q_{46} = (q_4 \hat{q_5} \hat{q_6})$ であり，$A_{13} = (a_1 a_2 a_3)$, $A_{46} = (a_4 a_5 a_6)$ である．また，$\hat{q_5}$ および $\hat{q_6}$ は式 (8) によって部分的に求まっている q_5 および q_6 の値である．

上記の変換によって，q_4, q_5, q_6 の計算における 12 回のベクトル行列積が，2 回の行列積と 3 回のベクトル行列積に変換できることが分かる．BLAS 3 アルゴリズムである行列積 (GEMM) は，BLAS 2 アルゴリズムである行列ベクトル積 (GEMV) に比べて，キャッシュの再利用性が高く，高い性能を発揮できる．

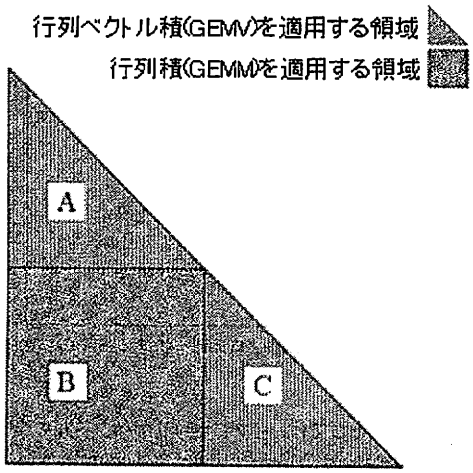

図 3 従来の手法における計算空間の分割方法

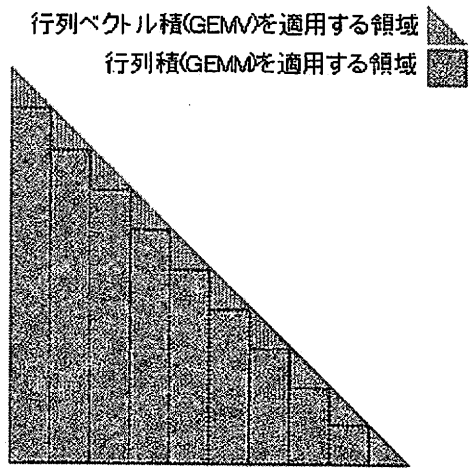

図 4 列方向ブロッキングによる CGS 法における計算空間の分割方法

文献 4) に示されている手法を m 行 n 列の行列 $A = (a_1 a_2 \cdots a_n)$ 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 3 に示す．図 3 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 3 においては，B 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，A および C 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，B で示されている範囲は BLAS 3 アルゴリズムである行列積（GEMM）で計算することができるが，A および C で示されている範囲は BLAS 2 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，BLAS 3 アルゴリズム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．

3.2 CGS 法における行列積の適用範囲を増加させる手法

3.1 節で述べた手法は，CGS 法においてベクトル $q_1, q_2, \cdots, q_j$ が既に直交化されているとき，$a_{j+1}, a_{j+2}, \cdots, a_n$ の直交化において，$q_1, q_2, \cdots, q_j$ 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 4) 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，$(n/2) \times (n/2)$ の正方行列に対して行列積を適用するのではなく，図 4 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ことである．この場合，行列ベクトル積で計算する部分は図 4 において小さい三角形で示される領域となり，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，列方向ブロッキングと呼ぶ．列方向ブロッキングによる CGS 法の疑似コードを図 5 に示す．なお，図 5 において，NB は一度に直交化するベクトルの本数である．

```
begin blockCGS(A, Q, n)
  q_1 = q_1/||q_1||
  do j = 2, n − NB, NB
    do i = j + 1, j + NB
      GEMV(Q^T_{j,i}, a_i, w);
      GEMV(Q_{j,i}, w, q_i);
      q_i = q_i/||q_i||
    end do
    GEMM(Q^T_{j+NB,n}, A_{j+NB,n}, S);
    GEMM(S, Q_{j+NB,n}, Q_{j+NB,n});
  end do
end
```

図 5 列方向ブロッキングによる CGS 法

3.3 提案する手法

本論文では，CGS 法における行列積の適用範囲を増加させる手法として，行列積を再帰的に適用する手法を提案する．図 3 で示されている，A および C 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 6 に示す A_B および C_B の領域においても $(n/4) \times (n/4)$ 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ブロックサイズまで小さくすることで，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

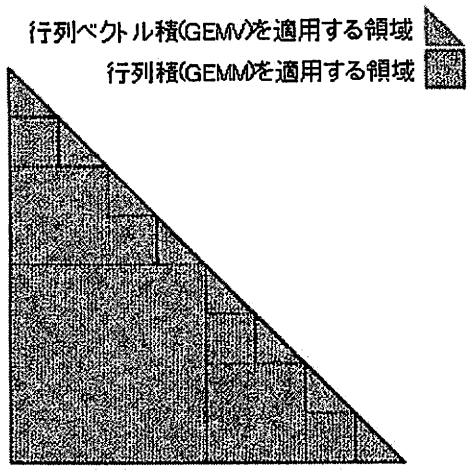

図 6 再帰的 CGS 法における計算空間の分割方法

```
begin recursiveCGS(A, Q, n, s, h)
  if (h <= NB) then
    q_s = q_s/||q_s||
    do j = s+1, s+h
      do i = j+1, j+h
        GEMV(Q^T_{j,i}, a_i, w);
        GEMV(Q_{j,i}, w, q_i);
        q_i = q_i/||q_i||
      end do
    end do
  else
    recursiveCGS(A, Q, n, s, h/2);
    GEMM(Q^T_{s,s+h/2}, A_{h/2+1,h}, S);
    GEMM(S, Q_{s,s+h/2}, Q_{h/2+1,h});
    recursiveCGS(A, Q, n, s+h/2, h/2);
  end if
end
```

図 7 再帰的 CGS 法

ズムを以後，再帰的 CGS 法と呼ぶ．再帰的 CGS 法の疑似コードを図 7 に示す．

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は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，ベクトル a_j，q_j および直交化が既に済んでいるベクトル $q_1, q_2, \cdots, q_{j-1}$ の各要素がそれぞれ別々のプロセッサに分散されている．したがって，ベクトルの内積を求める際には，リダクション演算（MPI では MPI_ALLREDUCE）を用いて行う必要がある[7]．

```
begin parallel_blockCGS(A, Q, n)
  q_1 = q_1/||q_1||
  do j = 2, n - NB, NB
    do i = j+1, j+NB
      GEMV(Q^{T_local}_{j,i}, a^local_i, w^local);
      MPI_ALLREDUCE(w^local, w, MPI_SUM);
      GEMV(Q^local_{j,i}, w, q^local_i);
      q_i = q_i/||q_i||
    end do
    GEMM(Q^{T_local}_{j+NB,n}, A^local_{j+NB,n}, S^local);
    MPI_ALLREDUCE(S^local, S, MPI_SUM);
    GEMM(S^local, Q^local_{j+NB,n}, Q^local_{j+NB,n});
  end do
end
```

図 8 行方向分散を用いた列方向ブロッキングによる並列 CGS 法

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[6]．

4.2 行方向分散を用いた並列 CGS 法

行方向分散を用いた並列 CGS 法は，図 8 のように表される．図 8 において，a_i^{local} および q_i^{local} は，各プロセッサに分散されているベクトル a_i および q_i を示している．

4.3 計算量および通信量

ここでは，図 8 で示すような，行方向分散を用いた列方向ブロッキングによる並列 CGS 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．一度に直交化するベクトルの本数を NB 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．

4.3.1 計 算 量

並列 CGS 法の計算量は，

- 内積を計算している前半部分では n^3 回
- ベクトル変換を計算している後半部分では n^3 回

となり，合計 $2n^3$ 回となる．

4.3.2 通 信 量

並列 CGS 法の通信量は以下のようになる．

- 図 8 における，GEMM と GEMM の間の MPI_ALLREDUCE において，1 回当たりの最小通信量は 8×NB（バイト），最大通信量は $8 \times n \times$NB（バイト）であるので，平均通信量は $8 \times \frac{n \times \mathrm{NB}}{2} = 4 \cdot n \cdot$NB（バイト）と仮定する．この通信が $n/$NB 回呼ばれるので，通信量は $(n/\mathrm{NB}) \times (4 \cdot n \cdot \mathrm{NB}) \cdot \log_2 P = 4 \cdot n^2 \cdot \log_2 P$(バイト）となる．
- 図 8 における，GEMV と GEMV の間の MPI_ALLREDUCE において，1 回当たりの最小通信量は 8（バイト），

表 1 単一プロセッサの評価環境

Platform	dual-core Xeon PC
Number of CPUs	2 cores, 4 CPUs
CPU Type	dual-core Xeon 2.8 GHz
L1 Cache	I-Cache: 12 K uops D-Cache: 16 KB
L2 Cache	2 MB
Main Memory	DDR2-SDRAM 2 GB
OS	Linux 2.6.9-5.ELsmp

表 2 PC クラスタの評価環境

Platform	dual Xeon PC SMP cluster
Number of Nodes	8
CPU Type	dual Xeon 2.4 GHz
L1 Cache	I-Cache: 12 K uops D-Cache: 8 KB
L2 Cache	512 KB
Main Memory	DDR-SDRAM 1 GB
OS	Linux 2.6.9-34.ELsmp

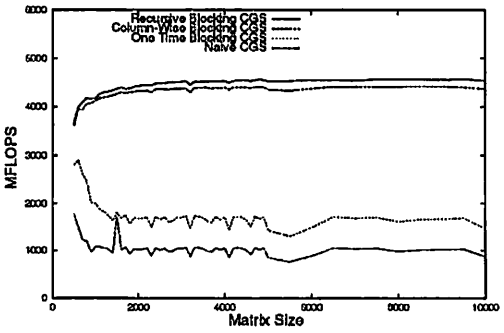

図 9 CGS 直交化の性能 (dual-core Xeon 2.8 GHz, 1 CPU)

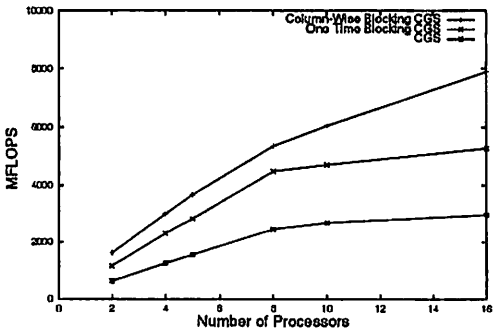

図 10 並列 CGS 法の性能 ($n = 8000$, dual Xeon 2.4 GHz PC SMP クラスタ)

最大通信量は 8×NB (バイト) であるので，平均通信量は $8 \times \frac{NB}{2} = 4 \cdot NB$ (バイト) と仮定する．この通信が n 回呼ばれるので，通信量は $(4 \cdot n \cdot NB) \cdot \log_2 P = 4 \cdot n \cdot NB \cdot \log_2 P$(バイト) となるが，$n \gg NB$ と仮定すると，この通信量はほぼ無視できる．

5. 性能評価

5.1 単一プロセッサにおける性能評価

単一プロセッサにおける CGS 法の性能評価にあたっては，BLAS 2 アルゴリズムである行列ベクトル積 (GEMV) を用いた naive な実装，従来の手法，列方向ブロッキングによる CGS 法，そして再帰的 CGS 法の性能を比較した．

行列サイズを $n = 500$ から $n = 10000$ まで変化させ，その平均の経過時間を測定した．なお，直交化の計算は倍精度実数で行っている．評価環境を表 1 に示す．

プログラムは C で記述した．コンパイラは Intel C Compiler (icc, Version 8.0) を用い，最適化オプションとして "icc -O3 -xP" を用いた．BLAS には Intel MKL 8.0 を用いた．

BLAS 2 アルゴリズムである行列ベクトル積 (GEMV) を用いた naive な実装，従来の手法，列方向ブロッキング，そして再帰的 CGS 法の各手法を用いた場合の，CGS 直交化の性能を図 9 に示す．

図 9 から，再帰的 CGS 法が最も高速であることが分かる．特に，$n = 10000$ の場合には再帰的 CGS 法は列方向ブロッキングに比べて約 4%高速であり，従来の手法に比べて約 3.1 倍，naive な実装に比べて約 5.3 倍高速になっている．

この理由としては，

- 再帰的 CGS 法は正方行列に対する行列積を行っているため，細長い長方形に対する行列積を行っている列方向ブロッキングに比べて行列積のカーネル部分の性能を高くできる．
- 再帰的 CGS 法や列方向ブロッキングは，従来の手法や naive な実装に比べて行列積を適用できる範囲が広く，キャッシュの再利用性が高いため，メモリアクセスが削減されている．

からであると考えられる．

なお，再帰的 CGS 法を用いて $n = 10000$ で 4532 MFLOPS (ピーク性能の約 81%) の性能が得られた．

5.2 PC クラスタにおける性能評価

PC クラスタにおける CGS 法の性能評価にあたっては，BLAS 2 アルゴリズムである GEMV を用いた naive な実装，従来の手法，そして列方向ブロッキングによる CGS 法の性能を比較した．

行列サイズを $n = 1000, 4000, 8000$ と変化させ，その平均の経過時間を測定した．なお，直交化の計算は倍精度実数で行っている．評価環境を表 2 に示す．

プログラムは C で記述した．コンパイラは gcc 3.4.5 を用い，最適化オプションとして "gcc -O3" を用いた．

通信ライブラリとしては，LAM MPI 7.0.6，BLASにはIntel MKL 8.0を用いた．SMPクラスタの各ノードは，1000Base-T の Gigabit Ethernet で接続されている．

BLAS 2 アルゴリズムである GEMV を用いた naive な実装，従来の手法，そして列方向ブロッキングの各手法を用いた場合の，CGS 直交化の性能を図 10 に示す．図 10 から，列方向ブロッキングが最も高速であることが分かる．特に，$n = 8000$，16 CPU の場合には列方向ブロッキングは従来の手法に比べて約 1.5 倍高速であり，naive な実装に比べると約 2.66 倍高速である．

この理由としては，列方向ブロッキングを行うことにより，CGS 法において BLAS 3 アルゴリズムである行列積（GEMM）を適用できる範囲が従来の手法に比べて広くなっていることから，キャッシュの再利用性が高く，メモリアクセスが削減されていることが考えられる．

また，スケーラビリティについてであるが，4.3 節で議論したように，並列 CGS 法では計算量が $2n^3$ であるのに対し，通信量が $4n^2 \log_2 P$ となる．したがって，LU 分解のように演算量が $O(n^3)$，通信量が $O(n^2)$ であるアルゴリズムに比べると $\log_2 P$ のオーダの分だけ通信量が多いために，プロセッサ数 P が多い場合には，高いスケーラビリティを確保するのは容易ではないことが分かる．

6. ま と め

本論文では，直交化アルゴリズムの一つである古典 Gram-Schmidt 法（CGS 法）の効率的な実装手法を提案した．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，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を増やすと共に，再帰的に行列積を行うことで，性能を改善することができることを示した．提案する手法を並列化して dual Xeon SMP クラスタに実装し，性能評価を行った．その結果，8 ノードの dual Xeon 2.4 GHz PC SMP クラスタでは $n = 8000$ において約 7.89 GFLOPS の性能を得ることができた．

今後の課題としては，再帰的 CGS 法の並列化および，最適なブロックサイズを決める手法の検討が挙げられる．

参 考 文 献

1) Björck, Å.: Solving least squares problems by Gram-Schmidt orthogonalization, *BIT*, pp. 1–21 (1967).
2) 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).
3) Björck, Å.: Numerics of Gram-Schmidt orthogonalization, *Linear Algebra Appl.*, Vol.197–198, pp. 297–316 (1994).
4) 寒川光: RISC 超高速化プログラミング技法, 共立出版 (1995).
5) Vanderstraeten, D.: A parallel block Gram-Schmidt algorithm with controlled loss of orthogonality, *Proc. Ninth SIAM Conference on Parallel Processing for Scientific Computing* (1999).
6) Katagiri, T.: *A Study on Large Scale Eigensolvers for Distributed Memory Parallel Machines*, PhD Thesis, the Department of Information Science, the University of Tokyo (2000).
7) 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, Vol. 2565, Springer-Verlag, pp. 302–314 (2003).
8) 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(ACS 11), pp. 279–288 (2005).