Vol.2012-HPC-133 No.26
2012/3/27

高精度行列 - 行列積アルゴリズムのスレッド並列化と ABCLibScript への機能実装

片桐孝洋[†]　尾崎克久[††]　荻田武史[†††]　大石進一[††††]

行列-行列積に代表される基本線形計算を集約したライブラリ BLAS (Basic Linear Algebra Subprograms) は，多くの線形計算で必須の処理である．従来の数値計算ライブラリは，演算速度は考慮しているが演算精度の考慮が不十分であり，解の精度保証が重要な課題となっている．本研究では，大石グループで開発された高精度行列-行列演算に 2 種のスレッド並列化を行った．予備評価の結果，並列処理の規模に応じ並列化方式を切り替える必要があることが判明した．また，その切り替えを実現できる自動チューニング（AT）を，AT 言語の ABCLibScript を用いて実現した．T2K オープンスパコン（1 ノード，16 スレッド）を用いた性能評価の結果，AT による並列化方式の切り替えで最大で 5 倍程度の速度向上を確認した．

A Thread Parallelization to Accuracy Guaranteed Matrix-Matrix Multiplication and Implementation of Its Function to ABCLibScript

TAKAHIRO KATAGIRI[†]　KATSUHISA OZAKI[††]
TAKESHI OGITA[†††] and SHIN'ICHI OISHI[††††]

BLAS (Basic Linear Algebra Subprograms), including matrix-matrix multiplication, is a crucial numerical library for many linear algebra computations. However, conventional numerical libraries are not enough taking into account for computing accuracy, while they are optimized for execution speed. Guaranteeing computational accuracy is one of important topics. In this research we parallelize an accuracy guaranteed matrix-matrix multiplication algorithm proposed by Oishi group by utilizing two kinds of thread implementations. As a result of preliminary evaluation, we found that selecting parallel method according to the number of threads is critical. In addition we adapt an auto-tuning (AT) language to establish the selection by using ABCLibScript. As a result of performance evaluation on the T2K open supercomputer (1 node, 16 threads) , we obtained maximum 5x speedup by using the AT.

1. はじめに

行列-行列積に代表される基本線形計算を集約したライブラリ BLAS (Basic Linear Algebra Subprograms)は，多くの線形計算で必須の処理である．一般に BLAS を含む従来の数値計算ライブラリでは演算速度は考慮しているが，計算結果の正確性の考慮が不十分なことが多い．解の精度保証が重要な課題となっている．一方で，BLAS を用いた汎用的な数値計算ライブラリ，たとえば，LAPACK において，精度保証をする研究や実装提案は，必ずしも多くはない．

BLAS を精度保証する研究が，早稲田大学の大石教授のグループにより進められている．本研究は大石グループで開発された高精度行列-行列演算（以降，**尾崎の方法**[1][2]と呼ぶ）を基本とし，その並列化を中心する以下の研究開発を行う．

i. 「疎行列 - 密行列積」，もしくは，「疎行列 - 疎行列」の実装方式の開発
ii. スレッド並列化手法の開発
iii. 分散メモリ型計算機による並列化手法の開発

本稿では，予備評価として，尾崎の方法に対して初期段階で行った，ii のスレッド並列化手法の結果について報告するものである．

本稿は以下の構成からなる．まず 2 節で，高精度行列-行列積アルゴリズムについて説明する．3 節は，スレッド並列化の説明と予備評価である．4 節は，3 節でおこなったスレッド並列化の自動チューニング（AT）を行う方式を，片桐が開発した AT 言語の ABCLibScript[3]に実装して性能評価を行う．5 節は，尾崎の方法の演算精度を評価する．6 節は，従来研究との比較である．最後にまとめを行う．

2. 高精度行列 - 行列積アルゴリズム

2.1 概要

尾崎の方法は，入力行列に対して，以下の無誤差変換を行う：

I)行列 A と行列 B を下記のように分解する(インデックスが若いほうが高いビット

† 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo.
†† 芝浦工業大学 システム理工学部
College of System Engineering and Science, Shibaura Institute of Technology.
††† 東京女子大学 現代教養学部
Division of Mathematical Science, Tokyo Woman's Christian University.
†††† 早稲田大学 理工学術院
Faculty of Science and Engineering, Waseda University.

を持つようにする）

$$A = A^{(1)} + A^{(2)} + A^{(3)} + \cdots + A^{(p)}$$
$$B = B^{(1)} + B^{(2)} + B^{(3)} + \cdots + B^{(q)}$$

II)行列積 AB を以下のように計算する（ $p \times q$ 個の行列積となる）

$$\begin{aligned} AB &= (A^{(1)} + A^{(2)} + \cdots + A^{(p)})(B^{(1)} + B^{(2)} + \cdots + B^{(q)}) \\ &= (A^{(1)} + A^{(2)} + \cdots + A^{(p)})(B^{(1)} + B^{(2)} + \cdots + B^{(q)}) \\ &= A^{(1)} B^{(1)} + A^{(1)} B^{(2)} + A^{(2)} B^{(1)} + \cdots + A^{(p)} B^{(q)} \end{aligned} \quad \cdots (1)$$

処理 I）の分解の仕方を工夫することで，処理 II)における行列積を無誤差の演算にすることができる[1]．行列サイズが大きくなるに従い，ほとんどの演算時間は行列積処理部分となる．

また，分解数である p， q を限定すれば，部分的な多倍長化演算と同様の効果を得ることができる．したがって，高精度化を図ることが可能である．

処理 I)の分解の過程で，入力行列の要素値のレンジに依存し，疎行列が生成される．このため密行列を入力としても，分解の過程で疎度が大きくなる場合は，密行列から疎行列化したほうがよい[1]．すなわち，「疎行列 - 密行列積」，および，「疎行列 - 疎行列積」の演算に切り替えるほうが，全体の計算量（実行時間）の縮減が期待できる．ただし，一般の数値計算ライブラリは，「疎行列 - 密行列積」，および，「疎行列 - 疎行列積」に特化した実装は提供されていないので，新規開発が必要である．また，この並列化は単純でないことが予想されるので，コードの独自並列化が必要となる．

尾崎の方法を組み込んだ数値計算ライブラリ全体でみると，単に実行速度のチューニングだけではなく，演算精度を考慮したチューニングも可能となる点にも新規性が期待される．

2.2 表記法

表記法について，以下にまとめる．

- 表記の説明

fl（・）： 最近点への丸めで計算
F： 浮動小数点数の集合
A： サイズが m 行 n 列の行列
B： サイズが n 行 p 列の行列
u ： the unit round off
2^{-24} ： IEEE 754 binary32 (single precision)
2^{-53} ： IEEE 754 binary64 (double precision)

2.3 尾崎の方法の主演算

尾崎の方法では，図１が主演算をなす．

```
Function F = EFT_Mul(A, B)
    [A, n_A] := Split_A;   [B, n_B] := Split_B;
    k := 1;
    for i =1: n_A
        for j =1: n_B
            F{ k } := A{ i } * B{ j };     k := k + 1;
end;   end;   end
```

図 1 $AB = \sum_{k=1}^{n_A \cdot n_B} F^{(k)}$ の変形部分

ここで図１の F の和には，faithful と呼ばれる結果を得るアルゴリズム[4]を用いている．そのため演算結果は，ほぼ最良になることが知られている．

図２に行列 A の分解部分の詳細を載せる．行列 B の分解については，図２の行列 A の分解に対し，転置した処理と同等なので説明を省略する．

3. 予備評価

3.1 目的

式(1)の部分行列のスレッド並列化には，以下の２通りの実現方法がある．そこで本評価では，この２種の性能評価を行うことが主な目的である．

I. 一つの行列 - 行列積を行う BLAS 関数 dgemm をスレッド並列化する方法
（**dgemm スレッド並列化**）
II. dgemm は逐次計算を行い，式（1）自体の並列性を抽出し，スレッド並列化する方法
（**逐次 dgemm で問題レベル並列化**）

```
Function [D, n ] = Split_A(A)
                  A
    n  := 0; n := size(A, 2)
     A
    while (norm (A, inf )~=0)
        n  := n  + 1;
         A     A
        μ  := max(abs(A),[ ],2);
                                  -1
        τ  := 2.^ceil ((log2(u  ) + log2(n +1))/2);
        t  := 2.^ceil((log2(μ)* τ ;
                  (nA)
        δ         := repmat(t, 1, q);
         A

                                  (nA)          (nA)
        A{n } := fl((A +  δ       ) -  δ        );
         A                 A           A

        A := fl(A  -  A{n });
                      A
end; end
```

図２　行列Aの分解部分の詳細

3.2　評価環境

東京大学情報基盤センターに設置された T2K オープンスパコン(東大版)(1 ノード, 16 コア）を利用し，スレッド並列化による並列化を行った．

日立製作所による C コンパイラ（日立最適化 C）を使用し，OpenMP 並列化を行った．最適化オプションは，“-Os -omp”である．また，GOTO BLAS ver.1.26 （スレッド並列版，および逐次版の双方）を BLAS 演算部分には利用している．

3.3　試験行列

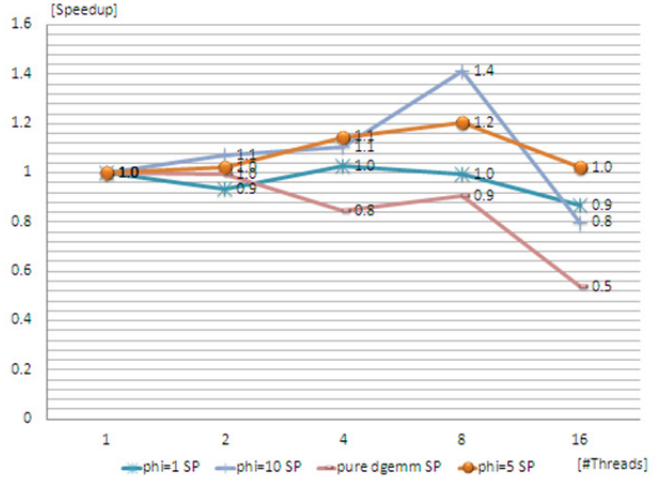
試験行列は，以下である．

- A, Bの要素を pow(10,rand()%Φ)　で生成

Φの値が大きいと，行列要素の値の分散が大きくなる．この場合，尾崎の方法による行列分割の回数が増え，行列 - 行列積の演算回数も増える．したがって，総合的な演算量（演算時間）が増加するので，並列処理の効果に影響する．

3.4　dgemm スレッド並列化の台数効果

図３に，dgemm スレッド並列化の台数効果を載せる．



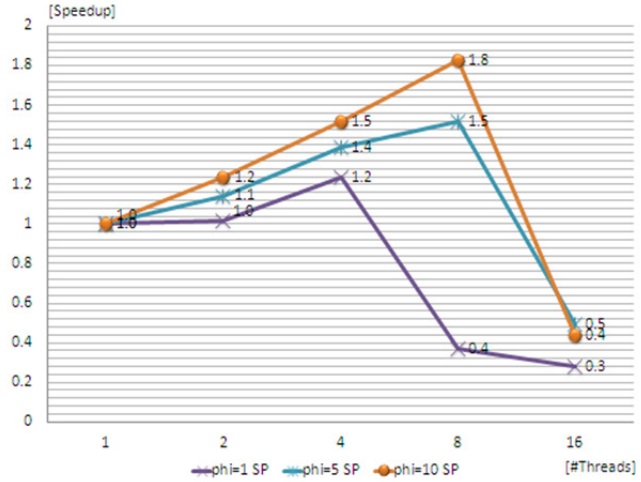
(a) n = 100（小規模問題）

(b) n = 8000（大規模問題）

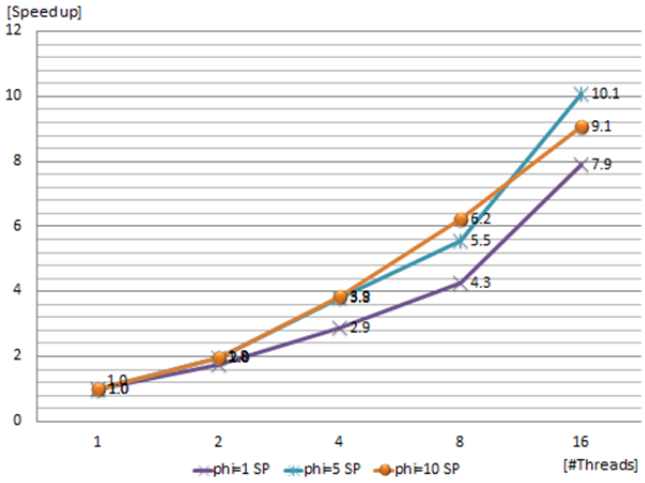
図３　dgemm スレッド並列化の台数効果

3.5 逐次 dgemm で問題レベル並列化の台数効果

図４に，逐次 dgemm で問題レベル並列化の台数効果を載せる．



(a) n = 100（小規模問題）



(b) n = 8000（大規模問題）

図４　逐次 dgemm で問題レベルの並列化の台数効果

3.6 考察

図３，図４より，以下の傾向が確認できる．

- 小規模問題では＜逐次 dgemm で問題レベル並列化＞が高速である．一方，大規模問題では，8 スレッドまでは＜dgemm スレッド並列化＞が高速である．
- 16 スレッドでは＜逐次 dgemm で問題レベル並列化＞が高速である．
- ＜dgemm スレッド並列化＞では，行列の分割数（Φの値）は並列性能にあまり影響しない．一方，＜逐次 dgemm で問題レベル並列化＞は，行列の分割数が多い（Φの値が大きい）時，並列性能がよい（特に，小規模問題）．

以上の考察から，どのスレッド並列化方式が有効となるかは，

- 実行時に決定するスレッド数と問題サイズ，および，問題の性質

に依存する。事前にスレッド並列化方式を固定できないことがわかる．

実行時の並列数と問題規模・性質を取得し，適する並列化方式を決定するソフトウェア構成方式の＜ソフトウェア自動チューニング＞（AT）技術の導入による高性能化が，尾崎の方法のスレッド並列実装でも期待される．

4. 自動チューニング言語機能への展開

4.1 概要

尾崎の方法の並列実装に AT を導入するには，既存の AT 用計算機言語で，尾崎の方法を実現したプログラム（ライブラリ化した場合のカーネルコードなど）に AT 方式を記述することが最も簡単な実現方法である．著者の片桐は，AT 言語の 1 つである ABCLibScript を開発している．ABCLibScript による尾崎の方法の AT 化がもっとも容易な方法である．

ABCLibScript では，チューニング変数の定義，チューニング変数の値の自動調整，任意のループの展開などコード自動生成，アルゴリズム選択に使えるコード選択など，AT で必須となる基本機能について提供する計算機言語である．

ABCLibScript による AT の記述形式は，＜ディレクティブ＞形式である．ループのスレッド化のための言語 OpenMP と同様の形式である．ユーザプログラムに AT 機能のコメント行を挿入するだけで，任意のプログラムに AT 機能を実現できる．元の逐次・並列プログラムの実行をまったく阻害せず，必要に応じ AT 機能が付加できることが最大の特徴である．この特徴は，レガシーコードにも AT 機能が容易に実装できることを意味している．したがって，数値計算ライブラリ分野のように，膨大で，かつ，高性能なレガシーコードを有する分野でのソフトウェア開発者のプログラムにおいて，AT 機能を付加する場合の開発コスト削減に寄与できる．

ABCLibScript は，計算機言語ごとにプリプロセッサを提供している．

C 言語版の ABCLibScript プリプロセッサでは,

```
#pragma ABCLib … region start ～  #pragma ABCLib … region end
```

で囲まれた任意のプログラムの領域に AT を施すことができる. この指定後, 専用プリプロセッサにより, AT 用のコード(C 言語でユーザが読めるもの)が自動生成される.

4.2 適用例

図５に, **インストール時 AT**（ソフトウェアのインストール時に行う AT）機能を, 尾崎の方法のメインカーネルにおいて,複数の行列-行列積に施す実装の＜実コード＞に適用した例を示す.

本稿で実装するスレッド並列化の例は, アルゴリズム選択機能が必要である. これは, ABCLibScript の＜select 指定子＞で指定可能する. Select 指定子で選択されるプログラム対象は,

```
#pragma ABCLib select sub region start
～
#pragma ABCLib select sub region start
```

の間に記載する. 図５に, そのコードを示す.

図５では, Fortran で記述されている BLAS の静的ライブラリ（GOTO BLAS）の関数コールに関し,（１）スレッド並列化された関数 dgemm_(…)をコールする実装と,（２）逐次 BLAS 関数 dgemm_seq(…)を問題サイズの並列性を利用してコールしている実装, の２種を選択できる.

問題レベル並列性を利用するスレッド並列化実装では, OpenMP を利用しているので, OpenMP のディレクティブが混載する. ABCLibScript では, OpenMP など, 他言語のディレクティブと共存が可能である. この点も, ABCLibScript のメリットである.

ここでは説明を割愛するが, AT 時のスレッド数や問題サイズの指定が, 図５では記載されていない. これらは, ABCLibScript のシステム変数を用いて, その対象と範囲を指定できる.

たとえば, 問題サイズを 100 から 1000 まで 100 刻みで調べる, という指定が, 容易に設定可能である.

そこで本実装では, 問題サイズを固定した上で, スレッド数を 1 から 16 まで 1 づつ変化させるチューニングの事例を紹介する.

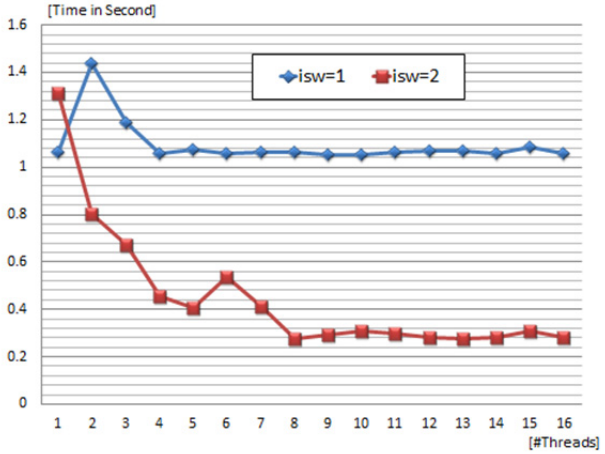
```
#pragma ABCLib install select region start
#pragma ABCLib name SelectOzaki
#pragma ABCLib select sub region start
  for (i=0;i<Ak;i++) {
    for (j=0;j<Bk;j++) {
      dgemm_(chn, chn, &m, &p, &n, &one,
        A+i*m*n, &lda, B+j*n*p, &ldb, &zero,
        C+m*p*(j+Bk*i), &ldc);
    } }
#pragma ABCLib select sub region end
#pragma ABCLib select sub region start
#pragma omp parallel for private(i,j)
  for (k=0;k<Ak*Bk;k++) {
    i = k / Bk; j = k % Bk;
    dgemm_seq(chn, chn, &m, &p, &n, &one,
       A+i*m*n, &lda, B+j*n*p, &ldb, &zero,
       C+m*p*(k), &ldc);  }
#pragma ABCLib select sub region end
#pragma ABCLib install select region end
```

図５　尾崎の方法のメインカーネルに対する ABCLibScript 記述の追加

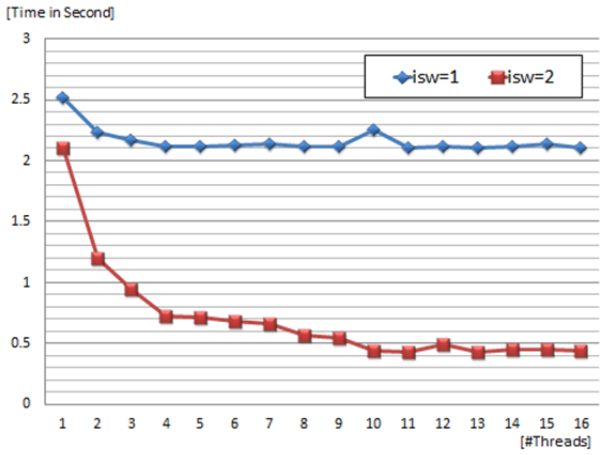
4.3 結果

図６に, ABCLibScript による自動チューニングの結果（実行時間[秒]）を載せる.

図７に, ABCLibScript による自動チューニングの効果（dgemm でスレッド並列化(isw=1)に対する速度向上）を載せる.

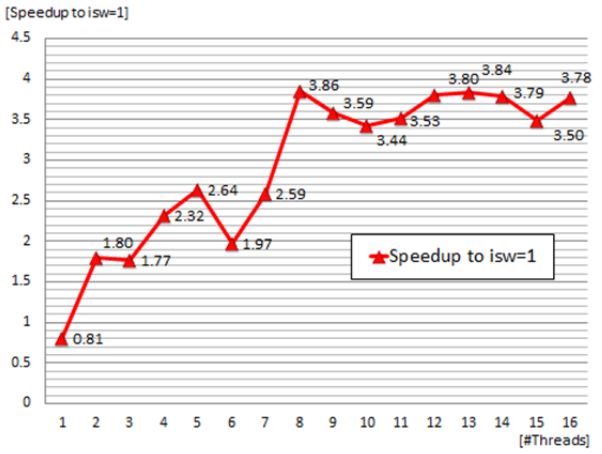


(a) n = 1000, Φ = 1 の結果

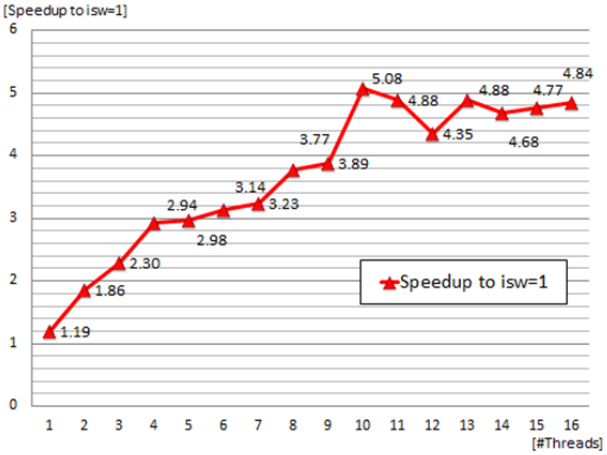


(b) n = 1000, Φ = 10 の結果

図 6　自動チューニングの効果（実行時間[秒]）．
isw=1 は dgemm スレッド並列化．isw=2 は逐次 dgemm で問題レベル並列化．



(a) n = 1000, Φ = 1 の結果



(b) n = 1000, Φ = 10 の結果

図 7　自動チューニングの効果（dgemm でスレッド並列化(isw=1)に対する速度向上）．isw=1 は dgemm スレッド並列化．isw=2 は逐次 dgemm で問題レベル並列化．

4.4 考察

図 6，7 から，n = 1000 では，

- 逐次 dgemm スレッド並列化（isw=1）が有効なのは，Φ = 1 でスレッド数 1 の時のみ．
- それ以外は，逐次 dgemm で問題レベル並列化(isw=2)が高速．
- 通常実現されると思われる逐次 dgemm スレッド並列化（isw=1）に対する AT の効果（速度向上）は，最大で 5 倍程度である．スレッド数が増えるほど一般に isw=2 が有効なので，AT の効果も大きい．

なお，問題の性質は実行時にならないと判明しないので，この例題の AT をインストール時に行うことはできない．したがって，問題を固定した後で AT を行う，実行起動前時 AT[3]に本機能を入れ込む方がよい．

実行起動前時 AT も，ABCLibScript で提供している．実行起動前時 AT を行うためには，ディレクティブ中の“install”を“static”に変更するだけでよい．

5. 精度評価

5.1 評価環境

ここでは，尾崎の方法の演算精度評価を行う．

評価に使用した CPU は，Xeon X5550，2.67 GHz(2 CPU，合計 8 core)であり，MATLAB 2011a を用いた．MATLAB 2011a における通常の倍精度演算の結果(A*B)と，尾崎の方法による演算結果の精度を比べる．

尾崎の方法は C のコードを MATLAB External Interfaces を利用して記述し，MATLAB 2011a に付属している libmwblas.lib を利用して dgemm を呼び出している．演算結果の行列の各要素について，真値との相対誤差を調べ，その最大を表にまとめる．

5.2 結果

表 1(a)に，3.3 節と同様の方法で行列要素を生成し，かつ行列要素の符号もランダムにつけた行列に対する，相対誤差の最大値を載せる．一方，表 1(b)は，表 1(a)の行列に対し逆行列を生成し，その生成した逆行列と，元の乱数行列の行列 - 行列積を行った結果をのせる．

表 1 と表 2 から，尾崎の方法を利用することで，通常の倍精度演算の dgemm では実現できない高精度演算ができる．

なお，通常の dgemm 演算時間や多倍長ライブラリを用いた演算時間と，尾崎の方法の演算時間や演算精度の比較については，文献[1]を参照のこと．

表 1　尾崎の方法と通常の行列 - 行列積の演算精度（相対精度の最大値）

(a)　乱数行列の積

乱数 / phi	1		5		10	
次元	近似	提案	近似	提案	近似	提案
10	8.83E-15	1.00E-16	6.79E-15	1.01E-16	1.97E-14	1.02E-16
100	5.21E-12	1.10E-16	1.08E-12	1.10E-16	1.30E-11	1.11E-16
1000	3.05E-09	1.11E-16	4.77E-09	1.11E-16	4.66E-09	1.11E-16
8000	2.03E-08	1.11E-16	1.96E-08	1.11E-16	2.37E-07	1.11E-16

(b)乱数行列と乱数行列から生成される逆行列との積

inv / phi	1	
次元	近似	提案
10	3.73E+02	1.10E-16
100	1.54E+03	1.11E-16
1000	6.83E+04	1.11E-16
8000	9.34E+05	1.11E-16

6. 関連研究

行列 - 行列積を含む，数値計算を高精度化（多倍長演算化）する研究は，多くの先行研究と開発ライブラリがある．

まず汎用的な多倍長演算ライブラリでは，GNU Multi-Precision Library (GMP) [5]や Extend precision floating-point arithmetic library (exflib) [6]が開発されており，多くの事例で利用されている．

数値計算ライブラリ LAPACK を多倍長化したライブラリとしては，Multiple precision arithmetic BLAS (MBLAS) and LAPACK (MLAPACK) [7]がある．

BLAS レベルの多倍長（混合精度）演算化を目的にしているものでは，Extra Precise Basic Linear Algebra Subroutines (XBLAS) [8]が知られている．XBLAS は，拡張精度として，double-double precision (128-bit total, 106-bit significand)を用いて実装されており，double-double precision の基本演算は +, -, *, / が開発されている．また混合精度演算では，いくつかの入出力において，異なる型（real と complex の混合，もしくは，精度（single と double）の混合，が提供されている。

本研究のアプローチとの違いは，尾崎の方法では，入力データの行列要素の値を考

慮し，必要な多倍長桁数が自動設定（動的設定）される方式になっている点である。上記のライブラリにおける桁数は，事前にユーザが設定（静的設定）する必要があるので，個別の演算に応じて，動的に多倍長演算の＜桁数＞を変更できない．すなわち，場合により無用に高精度演算がなされる可能性がある．

一方で尾崎の方法では，必要に応じて演算を疎行列化できることから，動的に計算量削減も実現できる点も，従来方式には無い特徴である．

7. おわりに

本稿では，高精度行列 - 行列演算を可能とする尾崎の方法について，スレッド並列化の実装方式 2 種を，T2K オープンスパコン（東大版）の１ノード（１６スレッド）を用いて評価した結果を紹介した．性能評価の結果，スレッド数，問題の性質に依存し，適するスレッド並列化の実装方式が変わることが明らかになった．

一方で，適するスレッド並列化の方式を自動チューニング（AT）するために，AT 言語 ABCLibScript に尾崎の方法でのスレッド並列化実装の選択機能を実現する記述と，実際にその AT 機能の組み込みを行った．性能評価の結果，素朴な BLAS レベルのスレッド並列化をする従来のスレッド並列化実装に対し，BLAS レベルで並列化をせず，問題レベルの並列性を利用する実装に切り替えることで，最大で 5 倍程度の高速化を実現できることが明らかになった．

以下は，今後の課題である．

- 今回の性能評価において実行性能が悪い，＜16 スレッド並列 dgemm＞の性能解析および改良法の提案．
- ccNUMA 構成特有の最適化（ファーストタッチ等）と，その AT 機能への取り込み．
- 尾崎の方法において，動的に切り替える「疎行列 - 疎行列積」用の特化ルーチンの開発．これにより，実行時の演算量削減を行い高速化を狙う．また，疎行列に切り替える条件の研究と，その AT 機能への取り込みも行う．
- MPI による分散メモリ並列化方式の開発．

本報告で示した尾崎の方法のメインカーネルの実装選択が可能な C 言語版 ABCLibScript のプリプロセッサのソースコードは，10 月 23 日リリース版として公開されている[9].

謝辞 本研究は，学際大規模情報基盤共同利用・共同研究拠点 H23 年度採択課題「高精度行列 - 行列積アルゴリズムにおける並列化手法の開発」(11-NA01)の支援による．

参考文献

1) Ozaki, K., Ogita, T., Oishi, S., Rump, S.M.: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numerical Algorithms, Vol. 59, No.1, pp.95-118, 2012.
2) 尾崎克久，荻田武史：浮動小数点数として最高の結果を返す行列積の計算法，京都大学 数理解析研究所 研究集会「科学技術計算における理論と応用の新展開」，2011 年
3) Katagiri, T.,Kise,K., Honda,H., and Yuba,T.: ABCLibScript: A Directive to Support Specification of An Auto-tuning Facility for Numerical Software, Parallel Computing, Vol. 32, No.1, pp.92-112, 2006.
4) Rump, S.M. , Ogita, T. , and Oishi, S.: Accurate floating-point summation part I: Faithful rounding. SIAM J. Sci. Comput., Vol. 31, No.1, pp.189-224, 2008.
5) The GNU Multi-Precision Arithmetic Library（GMP）
http://gmplib.org/
6) Extend precision floating-point arithmetic library
http://www-an.acs.i.kyoto-u.ac.jp/~fujiwara/exflib/
7) The MPACK; Multiple precision arithmetic BLAS (MBLAS) and LAPACK (MLAPACK)
http://mplapack.sourceforge.net/
8) Extra Precise Basic Linear Algebra Subroutines (XBLAS)
http://www.netlib.org/xblas/
9) ABCLibScript の C 言語版プリプロセッサに尾崎の方法の AT 機能を実現，2011 年 10 月 23 日リリース版，http://www.abc-lib.org/format.htm