Vol.2018-HPC-165 No.32
2018/8/1

GPGPU による高精度行列-行列積アルゴリズムのための Batched BLAS を用いた実装方式の提案

石黒史也[†1]　片桐孝洋[†2]　大島聡史[†3]　永井亨[†2]　荻野正雄[†2]

概要：基本線形代数サブプログラム BLAS は多くの線形計算で必須であるが，演算精度に関する考慮が不十分であることが多い．一方，倍精度演算による精度を保証する高精度行列-行列積アルゴリズムが知られている．また，複数の BLAS 演算をまとめて実行することで演算効率を向上させる，Batched BLAS という数値計算ライブラリが近年提案されている．本研究では，高精度行列-行列積アルゴリズムに Batched BLAS を適用した実装方式を提案するとともに，演算の途中で密行列から疎行列になる特性を利用した「疎行列-密行列」実装方式の演算性能について，東京大学に設置されている Reedbush-H システムの GPU 環境を利用して性能評価を行った．性能評価の結果，Batched BLAS を適用することで CPU 環境で最大 58.9%，GPU 環境で最大で 25.7%まで実行時間が短縮された．

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，多くの数値計算で必須である．しかし，BLAS を含む従来の数値計算ライブラリは，演算速度の向上は考慮しているが演算精度の向上に関する考慮が不十分であることが多い．つまり，情報落ち，桁落ちや丸め誤差などの影響で本来期待する精度の演算結果が得られないことがある．

一方, 尾崎が提案した高精度行列-行列積演算アルゴリズム（以降，**尾崎の方法**[1][2]）は，単精度や倍精度など利用している浮動小数点演算型の精度限界まで保証した高精度演算を行うことができる．本稿では尾崎の方法の実装に対して，同種の BLAS 演算を多数まとめて実行する Batched BLAS を適用した実装方式の提案を行う．また，Batched BLAS は CPU 環境だけでなく GPU 環境においても提供されているため， CPU 環境と GPU 環境の両方において Batched BLAS の有効性の検証を行う．

尾崎の方法では，以下の実装がなされている．

i. 「疎行列 - 密行列積」の実装方式
ii. 分散メモリ型計算機による並列化

本稿では，Batched BLAS の適用と性能検証に加え，「疎行列 - 密行列積」を GPU 環境で行ったものに対して，演算速度と演算精度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 節で尾崎の方法についての説明を行う．次に 3 節で Batched BLAS について説明し、尾崎の方法への適用に関して述べる．4 節は，「疎行列 - 密行列積」の実装方式の説明を記載する．5 節で Batched BLAS を適用した実装を，CPU 環境と GPU 環境で行った場合の性能と，「疎行列-密行列」実装方式の演算性能を，実行時間と演算精度の観点から評価する．最後に本稿のまとめを行い, さらにこれからの課題について述べる．

2. 高精度行列 - 行列積演算アルゴリズム

2.1 概要

以降，IEEE754 標準の浮動小数点型データを用いるものとする．

尾崎の方法は，入力行列に対して，以下の（1）～（2）の処理を行う．このとき，式(i)の無誤差変換を行う[1].

(1) **行列 A と行列 B を下のように分解する．右肩括弧内の数字が若い方が上位ビットを持つようにする．**

$$A = A^{(1)}+A^{(2)}+A^{(3)}+\ldots+A^{(p)}$$
$$B = B^{(1)}+B^{(2)}+B^{(3)}+\ldots+B^{(q)} \quad \ldots\text{(i)}$$

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

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

ここで，式(ⅱ)の分解された行列積どうしの加算には，高精度な和を行う演算で加算される[1].

処理(1)の分解の仕方を工夫することで，処理(2)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部分の時間となる．

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，演算時間を考慮して高精度化を図ることが可能である．

処理(1)の分解の過程で，入力行列の要素値の散らばり度合い（レンジ）に依存し，疎行列が生成される．こ

†1 名古屋大学 大学院情報学研究科
Graduate School of Informatics, Nagoya University
†2 名古屋大学 情報基盤センター
Information Technology Center, Nagoya University
†3 九州大学 情報基盤研究開発センター
Research Institute for Information Technology, Kyushu University

ⓒ 2018 Information Processing Society of Japan

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．

2.2 表記法

本稿で扱う表記法について，以下にまとめる．

- fl *(・)* : 最近点への丸めで行う浮動小数点演算
- ceil : 正の無限大方向へ丸めを行う関数
- repmat : 配列のコピーを行う関数
- 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 of 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

尾崎の方法の主要な演算部分を[1]を参照し，図 1 に MATLAB 風の疑似コード表記で示す．ここには式(i) および(ii)のうち，高精度和以外の部分が示されている．また行列 A の分解数を n_A，行列 B の分解数を n_B とした．

```
Function EF = 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
            EF{k} := A{ i } * B{ j };   k := k + 1;
end;   end;   end
```

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

図 1 の *EF* の和の結果は，真値に対して最も近い浮動小数点数か次に近い浮動小数点数のどちらかに必ず丸められることを保証する faithful な結果を保証する[3]．そのため，演算結果は，faithful な演算結果を保証するという観点で最良になることが知られている．

図 2 に，2.1 節の手順（1）の行列 A の分解部分を[1]を参照して MATLAB 風の疑似コード表記で載せる．この図 2 の変換処理のことを，**無誤差変換**と呼ぶ．行列 B の無誤差変換については，図 2 の行列 A の無誤差変換に対して転置した処理と同じであるため，説明を省略する．

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

図 2 行列 A の分解部分（無誤差変換）の詳細

3. Batched BLAS

3.1 概要

Batched BLAS[4]は，複数の行列の演算について，同種の BLAS 演算を並列に実行するインタフェースを提供する．ここではその中の GEMM 演算について記載する．このインタフェースにより従来の BLAS ライブラリにおいて計算資源を十分に使い切れない，小規模サイズの行列積を多数行う演算でも，まとめて計算することで，小さな GEMM 計算を単に連続して呼び出す性能に対して高い性能が期待できる．

例として，以下のような多数の組の GEMM 演算（行列-行列積演算)の並列計算を考える．ここで，A_i，B_iは多数の組の中の i 番目の入力行列であり，C_iは，それらの積である．

$$C_i \leftarrow \alpha_i A_i B_i + \beta_i C_i \quad i = 1 : N$$

Batched BLAS では，上記の行列サイズと，α_i，β_i の値を，一回の呼び出し（バッチ）全体で一定に保つか，可変にするかをアプリケーションに応じて，個別に設定することができる．したがって上記の並列計算を行う場合に，個別の GEMM の次元が小さくプロセッサ側の並列度が使い切れない場合でも全体としては十分使い切れると考えられる．

3.2 尾崎の方法への適用

本研究では，2.1 節の処理（2）における複数回の行列-行列積演算に対して，Batched BLAS を適用させることで，尾崎の方法において小規模な行列演算を多数実行する状況下での高性能化を狙う．この実装方式を，提案方式と呼ぶ．

ここで，本研究で使用した Batched BLAS の GEMM 演算のインタフェースについての説明をする．

以下の図 3，図 4 は，それぞれ CPU 環境と GPU 環境での実装で用いたインタフェースである．CPU 環境では Intel Math Kernel Library 2018.1.163[5]，GPU 環境では MAGMA ver.2.2.0 [6]に備えられた Batched 機能を用いた．図 3，図 4 を見るとわかるように，通常の GEMM 演算とは異なり入力行列がポインタの配列となる．また，

ⓒ 2018 Information Processing Society of Japan

group_count，または *batchCount* といった，GEMM には無い引数を必要とする．これらの引数変数として，入力行列に並列に計算したい行列の組をすべてポインタとして渡し，並列に計算したい行列の数を *group_count*，または *batchCount* で与えることが必要である．

```
void cblas_dgemm_batch
(    const CBLAS_LAYOUT Layout,
const CBLAS_TRANSPOSE* transa_array,
const CBLAS_TRANSPOSE* transb_array,
const MKL_INT* m_array, const MKL_INT* n_array,
const MKL_INT* k_array, const double* alpha_array,
const double **a_array, const MKL_INT* lda_array,
const double **b_array, const MKL_INT* ldb_array,
const double* beta_array, double **c_array,
const MKL_INT* ldc_array,
const MKL_INT group_count,
const MKL_INT* group_size);
```

図 3 Intel MKL における Batched BLAS の dgemm 演算インタフェース

```
void magmablas_dgemm_batch
( magma_trans_t transA,
magma_trans_t transB,
magma_int_t m,
magma_int_t n,
magma_int_t k,
double alpha,
double const*const* dA_array,
magma_int_t ldda,
double const*const* dB_array,
magma_int_t lddb
double beta,
double const*const* dC_array,
magma_int_t lddc,
magma_int_t batchCount,
magma_queue_t queue);
```

図 4 MAGMA における Batched BLAS の dgemm 演算インタフェース

以上を踏まえて，尾崎の方法への適用を行う．尾崎の方法の処理(2)における複数回の行列-行列積演算のプログラムを図 5 に表す．このプログラムでは，$p \times q$ の回数分，for ループで dgemm ルーチンを呼び出す．

図 5 に対し，Batched BLAS を適用させたプログラム（図 6）では，ポインタの配列に，行列 A と B の分割したそれぞれの先頭部分のポインタを，従来と同じ順番での計算となるように格納する．この収納によって結果を変えることなく，行列-行列積演算を並列に行うことができるようになる．

```
for (i=0;i<Ak;i++) {
        for (j=0;j<Bk;j++) {
            puremul(A+i*m*n,B+j*n*p,C+m*p*(j+Bk*i),
            lda,ldb,ldc);
        }
}
```

図 5 尾崎の方法の処理(2)部分のプログラム

```
j = 0;
for(i=1;i<=Ak*Bk;i++){
   a_array[i-1] = A+j*m*n;
   if(i % Bk == 0)
      j++;
}
j = 0;
for(i=1;i<=Ak*Bk;i++){
   b_array[i-1] = B+j*n*p;
   j++;
   if(i % Bk == 0)
      j=0;
}
for(i=0;i<Ak*Bk;i++){
   c_array[i] = C+i*m*p;
}
 cblas_dgemm_batch
        (CblasColMajor, chn, chn, m, p, n, one,
         a_array, lda, b_array, ldb, zero, c_array,
         m1, GRP_COUNT, size_per_group);
```

図 6 Batched BLAS を適用させた処理(2)部分のプログラム（提案方式）

4. 「疎行列 - 密行列積」の実装方式

4.1 概要

尾崎の方法に，「疎行列 - 密行列積」を行うため必要である疎行列格納形式を説明する．ここでは，CRS 形式について説明を行い，本稿では，疎行列化は CRS 形式の疎行列のすることとする．

4.2 CRS 形式

CRS（Compressed Row Storage）形式は，疎行列を行方向に順に走査し非零要素を格納する形式である．いま，$n \times n$ 行列 $A = (a_{ij})$ の非零要素数を nnz とする．このとき，CRS 形式では図 7 に示されるように，以下の 3 つの配列を使用する．

ⓒ 2018 Information Processing Society of Japan

1) 非零要素の値を保持する長さ nnz の配列 *val*
2) 配列 *val* に対応する非零要素の列番号を格納した，長さ nnz の配列 *colind*
3) 配列 *val*，*colind* における各行の開始位置を記憶する $n+1$ の配列 *rowptr*

$$\begin{bmatrix} a_1^1 & b_2^2 & c_3^3 & 0 \\ 0 & 0 & 0 & d_4^4 \\ e_1^5 & 0 & 0 & f_4^6 \\ 0 & 0 & g_3^7 & 0 \end{bmatrix}$$

$$\mathrm{val} = [a \quad b \quad c \quad d \quad e \quad f \quad g]$$
$$\mathrm{colind} = [1 \quad 2 \quad 3 \quad 4 \quad 1 \quad 4 \quad 3]$$
$$\mathrm{rowptr} = [1 \quad 4 \quad 5 \quad 7 \quad 8]$$

図 7 CRS 形式の例

ここで，SpMV の計算は，行ごとに独立して実行可能であるのに対し，CRS 形式が行単位で非ゼロ要素を格納する形式であるため，行単位の並列化が容易に実装できる．

4.3 並列化の実装方法

行列-行列積 *A B* を行う場合，行列の要素値の幅が大きい場合，無誤差変換により，行列 *A* と行列 *B* が疎行列になることが予想される．その場合，密行列演算を行うと多くが 0 演算となるため，無駄に計算量を増加させることになる．そこで，無誤差変換時に疎行列と判定される場合において，密行列から疎行列へ変換する実装を考える．

一般に，行列 *A* と行列 *B* で，双方とも疎行列になる可能性がある．しかし，行列-行列積演算 *A B* を疎行列-ベクトル積で行う場合の演算効率を考慮し，ここでは行列 *A* の疎度を判定して疎行列化することとする．

行列 *A* を疎行列化する場合，行列 *B* は密行列として扱う．このとき，行列-行列積 $C = A\,B$ は以下のようになる．

$$c_i = \mathrm{Sp}(A)\ b_i\ ,\ (i = 1,\ldots,n), \qquad \ldots(\text{iii})$$

ここで，行列 *C* の *i* 列目のベクトルを c_i，行列 *B* の *i* 列目のベクトルを b_i，および行列 *A* を疎行列化した疎行列を Sp(*A*)と記載した．ここで，式(iii)の計算は，疎行列-ベクトル積（Sparse Matrix-Vector Multiplications, SpMV）である．

式(iii)の計算をスレッド並列化する方法について，以下の 3 種類の実装方式が知られている[7].

(1) 内部並列化

SpMV 内でスレッド並列化する方法．SpMV の行単位の並列性を利用して，スレッド並列化する．（図 8）

(2) 外部並列化

SpMV 呼び出し部分でのスレッド並列性を使う．

(3) 複数右辺による内部並列化

複数右辺専用の SpMV における，内部のスレッド並列性を使う．

今回，CPU 環境での性能評価では(1)の内部並列化の実装を用いて行った．

```
#pragma omp parallel for private(s, j_ptr, i)
  for (i=0; i<n; i++) {
      s = 0.0;
      for (j_ptr=colind[i]; j_ptr<colind[i+1]; j_ptr++) {
        s = s + val[j_ptr] * b_i [colind[j_ptr]];   }
   ans[i] = s;
  }
```

図 8 内部並列化の OpenMP によるスレッド並列化のコード．ここで，*ans* は，計算結果，b_i は，行列 *B* の *i* 列ベクトルである．

また，本研究では尾崎の方法の「疎行列-密行列方式」実装を SpMV で行い，それを GPU 環境に適用させることも目的としている．そのため本研究では NVIDIA 社の GPU 向け疎行列計算ライブラリ cuSPARSE[10]を使用した．今回使用した，CRS 形式を用いた SpMV 演算のインタフェースを説明する．図 9 の csrValA，csrRowPtrA，csrColIndA にそれぞれ図 7 の val，rowptr，colind をポインタとして渡している．また，descrA で疎行列 *A* の種類や，CRS 形式のインデックスベースが 0 か 1 かを定めている．

```
cusparseDcsrmv
        (cusparseHandle_t    handle,
        cusparseOperation_t transA,
        int m, int n, int nnz, const double *alpha,
        const cusparseMatDescr_t descrA,
        const double *csrValA, const int *csrRowPtrA,
        const int *csrColIndA, const double *x,
        const double *beta, double *y)
```

図 9 cuSPARSE を用いた CRS 形式の SpMV 演算のインタフェース

5. 性能評価

5.1 評価環境

東京大学情報基盤センター設置のスーパーコンピュータ Reedbush-H[8]を利用する．表 1 にスペックを示す．

表 1 評価環境

マシン名	SGI Rackable C1102-GP8		
CPU		GPU	
プロセッサ名	Intel Xeon E5-2695v4 (Broadwell-EP)	プロセッサ名	NVIDIA Tesla P100 (Pascal)
プロセッサ数 (コア数)	2CPU(36)	コア数 (単体)	56

ⓒ 2018 Information Processing Society of Japan

周波数	2.1 GHz(Turbo boost 時最大 3.3 GHz)	メモリ容量（単体）	16 GB
理論演算性能	1209.6 GFlops/node	メモリ帯域幅（単体）	732 GB/sec
メモリ容量	256 GB/node	理論演算性能（単体）	5.3 TFlops
メモリ帯域幅	153.6 GB/sec/node	搭載数	2
コンパイラ	Intel C++ コンパイラ (ver.18.1.163)	コンパイラ	Intel C++ コンパイラ (ver.18.1.163)
オプション	-xHost -O3 –qopenmp –mkl	オプション	-xHost -O3 -fPIC -DNDEBUG -DADD_ -fp-model strict -qopenmp -lmkl_intel _lp64 -lmkl_intel _thread -lmkl_core -liomp5 –lpthread -lcublas -lcudart -lcusparse

5.2 入力行列

行列-行列積 $C = AB$ を計算する．

ここで試験行列として，A は，単位行列と，ある疎度分の要素に対して 0~1 の範囲で生成した値で置き換えた行列とする．B は A の逆行列とする．今回，疎度を 90%に設定した．また尾崎の方法によって分解された行列 A の疎度が，90%以上になる場合に疎行列-密行列形式（SpMV）を利用するようにした．

なお，本試験行列は行列 B が行列 A の逆行列であるため，行列積演算の中で演算のキャンセルが起こり，激しい丸め誤差を生じることが予想される．そのため，DGEMM 演算による結果では，精度が出ないことが本研究でも実験的に示されている．

行列サイズは，N＝10，50，100，500，1000，5000，10000 の 7 種類を利用した．Batched BLAS についての性能評価においては N=10~5000, 疎行列-密行列形式についての性能評価においては N=50~10000 を利用した．

5.3 実験設定

本研究では，すべての行列-行列積演算を，倍精度で行った．CPU 環境においては，1 ノードでスレッド数を 1 から 36 まで変化させて実行時間を測定した．Batched BLAS については，Intel Math Kernel Library 2018.1.163[5]に備えられた Batched 機能を用いた．今回は Intel Math Kernel Library の Batched 機能の中の関数 “cblas_dgemm_batch” を用いて評価を行った．

次に，GPU 環境では，実行環境として CUDA 8.0.44[9]を使用した．また CPU のスレッド数を 35 に固定して測定した．今回，35 スレッドを利用した理由は，基本的に実行時間はスレッド数を増やすと減少したが，36 スレッドの時だけ遅くなってしまったためである．Batched BLAS については，MAGMA ver.2.2.0 [6]の関数 ”magmablas_dgemm_batch” を用いて評価を行った．

cuSPARSE については，CUDA 8.0.44[9] の，関数”cusparseDcsrmv”を使用した．

CPU および GPU 双方の実行環境においても，実行時には NUMA Affinity と numactl の設定を行っている．具体的には, export KMP_AFFINITY=granularity=fine, compact, 1,0 と，numactl <実行コマンド> をジョブスクリプト中に記述した．また，実験ではすべての時間計測に関数”omp_get_wtime();” を用いた. ただし GPU を利用した部分の時間計測だけは ”omp_get_wtime();” の前に ”cudaThreadSynchronize();” を追加した．

5.4 Batched BLAS の性能評価

市村らのマルチコア向け実装[11]を基にし，尾崎の方法に Batched BLAS を適用させた場合の性能を実行時間と演算精度の観点から評価を行う。

5.4.1 無誤差変換における分割数

行列サイズごとの行列 A と行列 B に関する無誤差変換の分解数（それぞれ，Ak および Bk と記載）と，Ak のなかで疎行列化された数（Spm）を以下の表 2 に載せる．

表 2 試験行列における無誤差変換による行列の分割数

サイズ	*Ak*	*Bk*	*Spm*
10	2	3	0
50	3	3	3
100	3	3	3
500	3	4	3
1000	3	4	3
5000	3	5	3
10000	4	4	4

ⓒ 2018 Information Processing Society of Japan

5.4.2 演算精度の比較

尾崎の方法の演算精度の観点での優位性を示すため，

(1) **通常の** dgemm **を用いた場合（通常）**
(2) **尾崎の方法を用いた場合（尾崎）**
(3) **尾崎の方法と** Batched BLAS **を適用させた場合**（Batch）
(4) **尾崎の方法に疎行列-密行列形式を用いた場合**（CRS）

の演算精度を絶対誤差と相対誤差に分け，表 3 に載せる．

表 3　試験行列における演算精度

サイズ	10	50	100	500
通常(絶対)	8.77e-17	8.18e-16	1.70e-15	2.94e-15
通常(相対)	2.11e-10	1.82e-8	7.04e-7	2.71e-5
尾崎(絶対)	8.77e-17	1.11e-16	1.10e-16	1.11e-16
尾崎(相対)	8.77e-17	1.11e-16	1.10e-16	1.11e-16
Batch(絶対)	8.77e-16	1.11e-16	1.10e-16	1.11e-16
Batch(相対)	8.77e-17	1.11e-16	1.10e-16	1.11e-16
CRS(絶対)		1.11e-16	1.10e-16	1.11e-16
CRS(相対)		1.11e-16	1.10e-16	1.11e-16

サイズ	1000	5000	10000
通常(絶対)	5.33e-15	5.14e-15	
通常(相対)	1.92e-5	3.42	
尾崎(絶対)	1.11e-16	1.11e-16	
尾崎(相対)	1.11e-16	1.11e-16	
Batch(絶対)	1.11e-16	1.11e-16	
Batch(相対)	1.11e-16	1.11e-16	
CRS(絶対)	1.11e-16	1.11e-16	1.11e-16
CRS(相対)	1.11e-16	1.11e-16	1.11e-16

表 3 を見るとわかるように，通常の dgemm 演算の場合では行列のサイズを大きくすればするほど演算精度が破綻してしまうが，尾崎の方法を適用した場合，また尾崎の方法と Batched BLAS を適用した場合，疎行列-密行列形式の場合では精度が保証されていることが分かった．このことはいずれのサイズの通常(相対)の値が大きくなってしまっているのに対し，Batch(相対)や CRS(相対)の値が行列サイズを大きくさせても限りなく小さい値を維持していることからわかると考えられる．

5.4.3 ルーチン全体の演算速度の比較

CPU 環境と GPU 環境において，ルーチン全体の時間を載せる．ルーチン全体の時間とは，無誤差変換時間およびその他の時間を含む尾崎の方法の実装全体の実行時間である．ここでは，無誤差変換時間を error_free，DGEMM ルーチンの実行時間を kernel，尾崎の方法の実装におけるその他の部分を other_kernel とした．

結果を図 10 ～図 15 に示す．なお，従来とは DGEMM ルーチンを用いて複数行列積を行う従来法を指し，提案は Batched BLAS による実装の提案法を指す．また，CPU と GPU の実行時間の差が大きかったため，対数グラフであることに注意されたい．

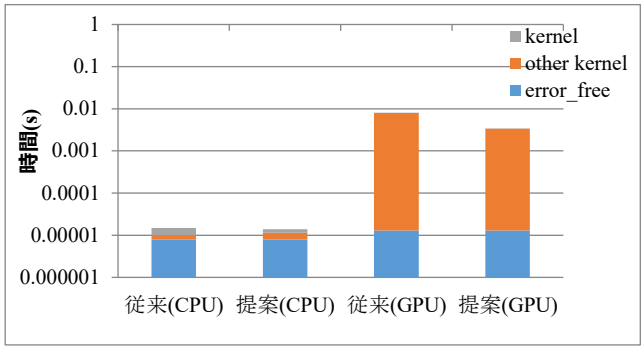


図 10　N=10 での CPU，GPU 環境のルーチン全体の演算時間

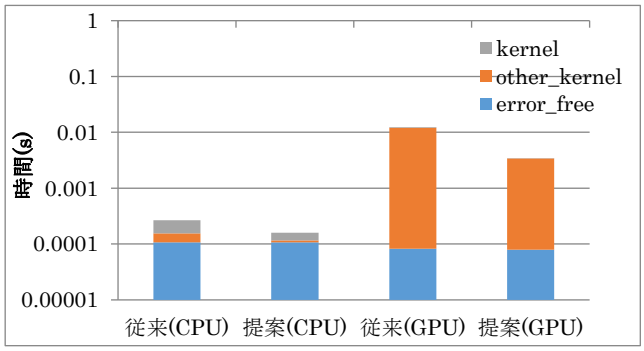


図 11　N=50 での CPU，GPU 環境のルーチン全体の演算時間

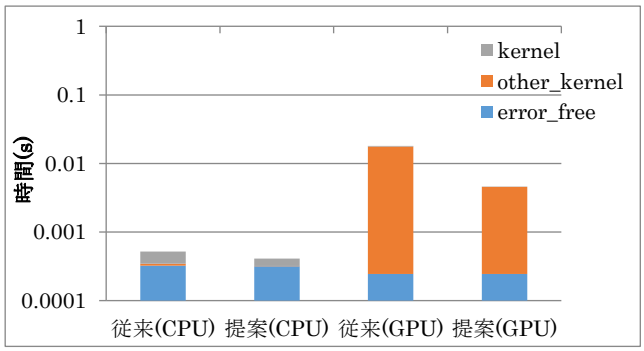


図 12　N=100 での CPU，GPU 環境のルーチン全体の演算時間

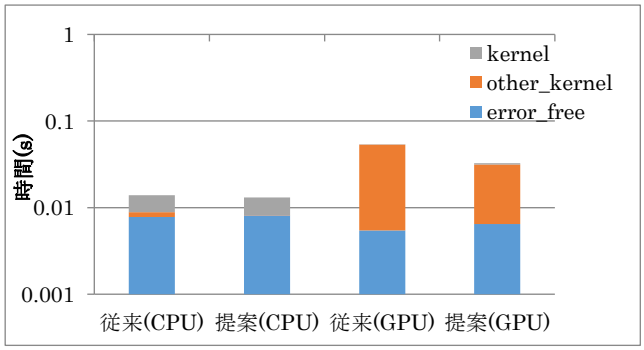


図 13　N=500 での CPU，GPU 環境のルーチン全体の演算時間

ⓒ 2018 Information Processing Society of Japan

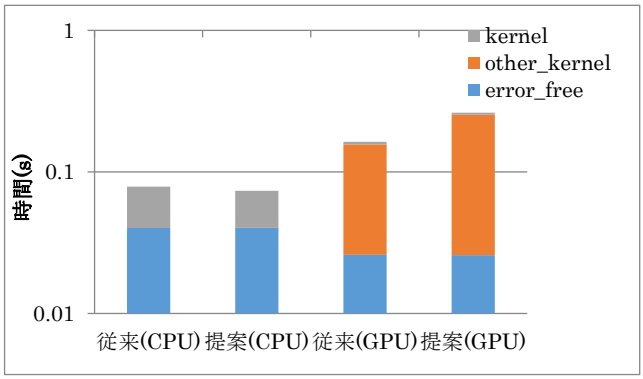


図 14　N=1000 での CPU，GPU 環境の
ルーチン全体の演算時間

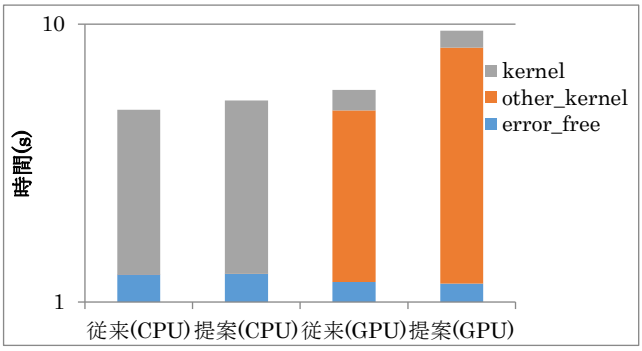


図 15　N=5000 での CPU，GPU 環境の
ルーチン全体の演算時間

図 10-図 15 から，提案手法を用いた場合，行列サイズが *N*=50 の時（図 11），CPU 環境で最大 58.9%まで，GPU 環境では，サイズが N=100 の時（図 12）に最大で 25.7%まで実行時間が短縮された．しかし，行列サイズが大きくなると CPU 環境，GPU 環境ともに提案手法の有用性があまりないということが分かった．また，GPU 環境での other_kernel の時間がかなり長くなっているのは，CPU-GPU 間のデータ転送時間を含んでいるからであり，実用上影響がある処理であると考えられる．

5.4.4 演算性能の比較

演算性能の観点から提案手法との比較を行う．図 16 に，CPU，GPU 環境での演算性能を示す．

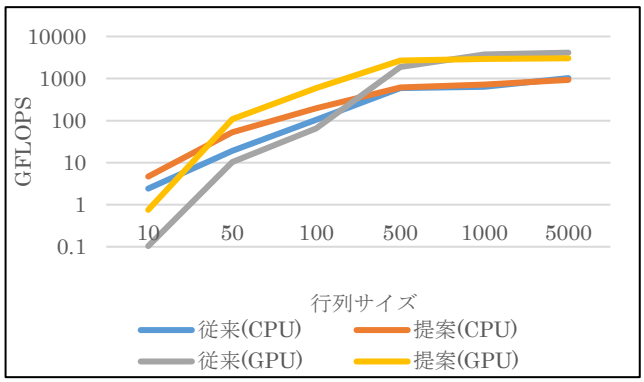


図 16　CPU，GPU 環境での演算性能

図 16 を見るとわかるように，行列サイズが大きくなると演算性能も向上することが分かった．また，行列サイズが大きいときは提案手法の演算性能は従来手法に比べ劣ってしまうことが分かった．N=5000 のときの従来手法では，CPU，GPU ともに Broadwell-EP，Pascal の理論性能に対して 8 割ほどの演算性能が出ているため，BLAS を用いた尾崎の方法の実装効率は妥当であると考えている．

5.5 「疎行列-密行列形式」における性能評価

5.5.1 ルーチン全体の演算速度の比較

疎行列-密行列形式において，CPU 環境と GPU 環境でのルーチン全体の演算時間を載せる．

ここでルーチン全体の時間とは，無誤差変換時間及び疎行列形式への変更時間の実際の計算時間である．無誤差変換時間を error_free, 疎行列の crs 形式への変換時間を setmat, GPU へのメモリ転送時間等も含めた SpMV 演算ルーチン時間を crs，その他の時間を other とした．

結果を図 17 に示す．サイズが小さいときと大きいときの実行時間の差が大きかったため，対数グラフであることに注意されたい．

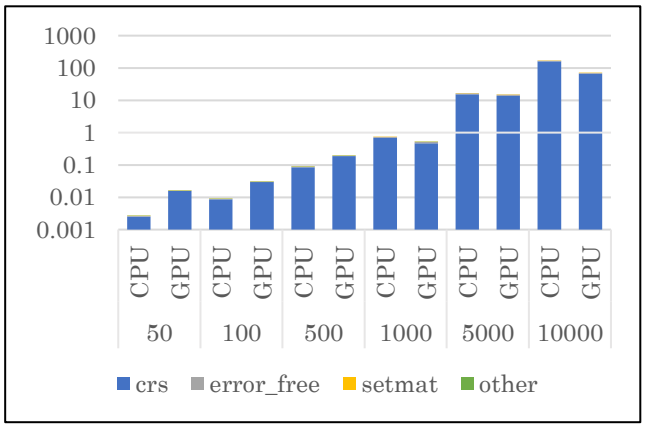


図 17　疎行列-密行列形式の
ルーチン全体の実行時間

図 17 から，行列サイズが 1000 より大きい場合では，GPU 環境の方が，ルーチン全体の演算時間が短いことが分かった．これは，行列サイズが小さいときには，GPU デバイスへのメモリ転送コストが計算時間に対して大きなものとなっていたが，行列サイズが大きくなることで，メモリ転送のコストが相対的に下がったからであると考えられる．今回，サイズが N=10000 の時，CPU 実行に対して、GPU 実行では，最大で 42.8%まで全体の実行時間が短縮された．

6. おわりに

本論文では，高精度行列-行列積演算のアルゴリズムである尾崎の方法で現れる複数の行列-行列演算に対して，特に小規模行列を多数行う場合において従来法である BLAS 実装（DGEMM 関数での実装）の演算効率が低下する問題に

ⓒ 2018 Information Processing Society of Japan

着目した．小規模行列を多数行う場合に高性能化が期待できるライブラリである Batched BLAS を適用することで，尾崎の方法を高速化する実装方式を提案した．

また，同時に尾崎の方法を適用させたときに行列が疎行列化する場合に行列-行列積を「疎行列-密行列方式」で計算を行う．その計算を，GPGPU を利用して行った．これらの提案法の有効性を評価するため，東京大学情報基盤センター設置の GPU クラスタ型のスーパーコンピュータ Reedbush-H 上の CPU および GPU を用いて性能評価を行った．

まず，Batched BLAS を適用させた場合であるが，ルーチン全体の演算時間の観点から，Batched BLAS を適用することで，CPU 環境で最大 58.9%，GPU 環境で 25.7%まで実行時間を短縮させることができた．その観点で，提案法の有効性が示された．この一方で，行列のサイズを大きくすると，CPU 環境でも GPU 環境でも提案法の実行時間は増大してしまうことが明らかとなった．

また演算性能を比較すると，行列のサイズが 500 以下の場合では CPU, GPU 環境のどちらにおいても Batched BLAS を適用させた提案法のほうが，演算性能が高い．しかし，行列サイズが大きくなると従来手法の方が，演算性能が高いということが明らかになった．

次に，疎行列-密行列形式を使用した場合の性能評価の結果は， GPU 環境では行列サイズを大きくした場合に演算速度が向上することを確認した．具体的には，全体の演算時間の観点で，行列サイズが N=10000 の時に，最大で 42.8%まで実行時間が短縮された．

今後の課題を以下に述べる．まず Batched BLAS を適用させた場合における，行列-行列積演算部分の性能チューニングがある．特に，GPU の利用では，行列サイズが小さい場合は，CPU で実行するほうが高速である．行列サイズと GPU の性能を事前に評価したうえで，実行時に CPU 実行か GPU 実行かを動的に選択する実装が有効になると思われる．この観点では，自動性能チューニング[12]の適用が有力である．そのため，自動チューニングの適用は重要な今後の課題である．

二つ目に，CPU 環境では，疎行列格納形式の一つである ELL 形式が「疎行列-密行列形式」ですでに実装されている[11]．しかし尾崎の方法の処理において我々は，GPU 環境での ELL 形式の実装評価を行っていない．よって，GPU 環境における ELL 形式についての追加実装をし，性能評価を行う必要がある．

謝辞

本研究の一部は，文部科学省委託事業，ポスト「京」萌芽的課題アプリケーション開発，萌芽的課題 1，基礎科学のフロンティアー極限への挑戦「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」，および科学技術研究費補助金，基盤研究（B）「通信回避・削減アルゴリズムのための自動チューニング技術の新展開」（課題番号：16H02823）の支援による．

参考文献

[1] K. Ozaki, T. Ogita, S. Oishi, S. M. Rump: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numarical 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] S. M. Rump, T. Ogita, and S. Oishi, : Accurate floating-point summation part Ⅰ: Faithful rounding. SIAM J. Sci. Comput., Vol. 31, No. 1, pp. 189-224, 2008.

[4] J. Dongarra, S. Hammarling, N. J. Higham, S. D. Relton, P. Valero-Lara, and M. Zounon, The Design and Performance of Batched BLAS on Modern High-Performance Computing Systems, Proc. of ICCS2017, pp. 495-504 (2017)

[5] Intel: Intel Math Kernel Library. https://software.intel.com/en-us/mkl (参照 2018-06-29)

[6] MAGMA Blas Webpage. http://icl.cs.utk.edu/magma/index.html (参照 2018-06-29)

[7] 片桐孝洋，尾崎克久，荻田武史，大石進一：高精度行列-行列積アルゴリズムの疎行列演算化による高速化，日本応用数理学会「行列・固有値問題の解法とその応用」研究部会第 15 回研究会，SWoPP2013, 2013 年

[8] Reedbush スーパーコンピューターシステム [東京大学情報基盤センタースーパーコンピューティング部門]. https://www.cc.u-tokyo.ac.jp/supercomputer/reedbush/service/, (参照 2018-06-29)

[9] CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit (参照 2018-06-29)

[10] cuSPARSE Toolkit. https://developer.nvidia.com/cusparse, (参照 2018-06-29)

[11]市村駿太郎，片桐孝洋，尾崎克久，荻田武史，永井亨，荻野正雄: マルチコア計算機による高精度行列 - 行列積アルゴリズムの性能評価，研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2017-HPC-160(16)，pp. 1-8 (2017).

[12] 弓場敏嗣，“数値計算応用を対象とする自動性能チューニング技術，” 電気通信大学紀要 20 巻 1・2 合併号，pp. 1-14，2007.

ⓒ 2018 Information Processing Society of Japan