2H-02

高精度行列-行列積アルゴリズムにおける Batched BLAS の適用

石黒 史也† 片桐 孝洋‡ 大島 聡史†3 永井 亨‡ 荻野 正雄‡

名古屋大学工学部電気電子・情報工学科†

名古屋大学 情報基盤センター‡

九州大学 情報基盤研究開発センター†3

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の処理である．しかし，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．

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）は，利用している浮動小数点演算型の精度限界まで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．性能評価では，Batched BLAS の有効性の検証を行う．

本稿は以下の構成からなる．まず 2 節で Batched BLAS についての説明と尾崎の方法への適用を行う．3 節で Batched BLAS を適用した実装を行った場合の性能を実行時間と演算精度の観点から評価する．最後にまとめを行う．

2. Batched BLAS と尾崎の方法

Batched BLAS[1]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．このことで，計算資源を十分に使い切れないサイズの BLAS 演算でもまとめて計算することで，小さな BLAS 計算を単に連続して行うよりも全体として高い性能が期待できる．

例えば，以下の一般的な行列-行列演算(gemm)の並列計算を考える．

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

この場合，行列サイズと，α_i, β_i の値を，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．

この Batched BLAS を尾崎の方法に適用させることを考える．尾崎の方法ではまず、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 q 個の行列積となる）

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

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．

本研究では，処理 II)における，複数回の行列-行列積演算に対して，Batched BLAS のインタフェースを適用させることで，小規模な行列演算を多数実行する状況下で高性能化を狙う．

3. 性能評価

ここでは，市村らのマルチコア向け実装[2]を基にし，尾崎の方法に Batched BLAS を適用させた実装の性能評価を行う．

Batched BLAS については，Intel Math Kernel Library 2018.1.163[3]に備えられた Batched 機能を用いる．

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．

An Adaptation of batched BLAS to algorithm of high precision matrix-matrix multiplications

† Fumiya Ishiguro, Electrical and Electric Engineering and Information Engineering, School of Engineering, Nagoya University

‡ Takahiro Katagiri, Toru Nagai, Masao Ogino, Information Technology Center, Nagoya University

†3 Satoshi Ohshima, Research Institute for Information Technology, Kyushu University

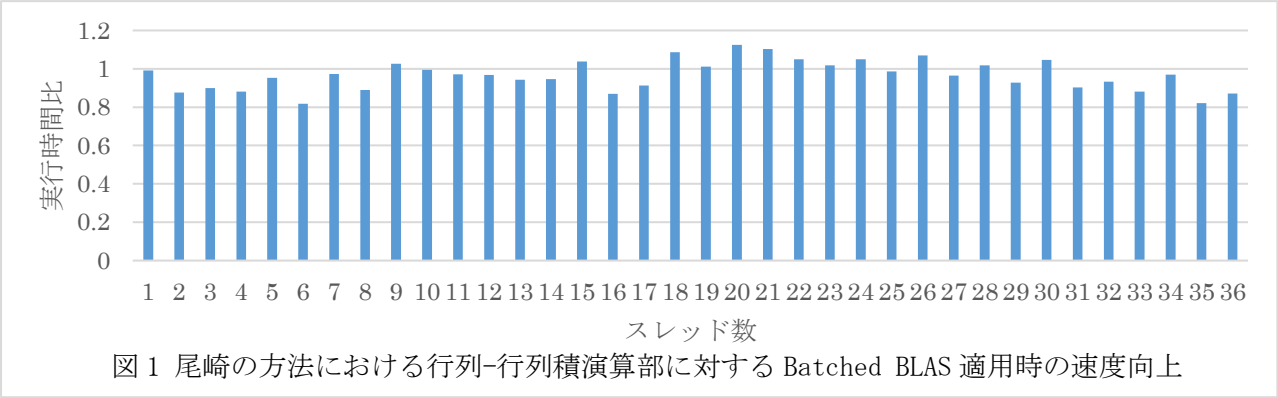

図 1 尾崎の方法における行列-行列積演算部に対する Batched BLAS 適用時の速度向上

3.1 評価環境

東京大学情報基盤センター設置の Reedbush-H を利用する．表 1 にスペックを示す。

表 1　評価環境

SGI Rackable C1102-GP8	
CPU: Intel Xeon E5-2695v4 (Broadwell-EP)	
プロセッサ数 (コア数)	2CPU (36 コア)
周波数	2.1GHz(Turbo boost 時 最大 3.3GHz)
理論演算性能	1209.6 GFlops / node
メモリ容量	256 GB / node
メモリ帯域幅	153.6 GB/sec / node
コンパイラ	Intel C++ コンパイラ (ver.18.1.163)
オプション	-xHost -O3 -qopenmp

3.2 実験条件

行列サイズは N=500 とする.利用ノード数は 1 ノード，スレッド数は，1 スレッドから 36 スレッドまで変化させる．

3.3 入力行列

$C = AB$ 計算時の行列 B は A の逆行列を代入．この時，尾崎の方法で分解される行列の数（式(1)における p と q）は，$p = 3$，$q = 4$ となる．

3.4 結果と考察

Reedbush-H における CPU 環境において Batched BLAS を適用させた場合の，尾崎の方法の行列 - 行列積演算部分の実行時間の評価を行う．図 1 に，Batched BLAS 適用前の時間を 1 とした時，Batched BLAS 適用時における実行時間の比を示す（値が小さいほど高速を意味する）．

図 1 から，スレッド数が 18-26 の間以外は Batched BLAS を適用すると高速化することがわかる．スレッド数が 6 の場合が最も速度向上し，実行時間が 81.7% に短縮された．

また，batched BLAS を適用しない場合の最短実行時間は 0.004951s(34 スレッド)で，適用した場合は 0.004622s(35 スレッド)であった．

図 1 において，18-26 スレッドで性能が出ない理由を考える．利用コア数（＝スレッド数）が 18 の周辺で生じているため，ソケットを跨ぐ実行時に生じている可能性がある．このことから，NUMA アクセスの影響を受けている可能性があるが，性能プロファイラによる詳細解析が必要である．

4. おわりに

本稿では，尾崎の方法による高精度行列-行列積演算において，Batched BLAS を適用した実装方式を提案した．東京大学情報基盤センター設置の Reedbush-H の CPU 環境を利用して性能評価を行った．性能評価の結果，一部のスレッド数を除いては速度向上が得られることが分かった．

今後の課題として，性能プロファイラによる性能解析，および GPU 環境での性能評価が挙げられる．

謝辞

本研究の一部は，JSPS 科研費 16K12432 の助成，および，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」の支援による．

参考文献

[1] 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)

[2] 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).

[3] Intel: Intel Math Kernel Library, https://software.intel.com/en-us/mkl (accessed 2018/1/11)