Vol.2013-HPC-138 No.6
2013/2/21

Strassen のアルゴリズムを付加した 行列積自動チューニングライブラリ

坂本真貴人[†1]　藤井昭宏[†1]　田中輝雄[†1]

行列行列積を計算する DGEMM の性能は，さまざまな科学技術計算において重要である．DGEMM の高速化の手法の 1 つに Strassen のアルゴリズムがある．これは再帰的アルゴリズムであり，適用する回数を増やすことで計算量を$O(N^3)$から$O(N^{\log 7})$まで削減することができる．しかし，計算機や行列サイズに合わせた適切な回数を選択しないと高速化できない．本研究では，Strassen のアルゴリズムを，自動チューニング機能付きの線形代数ライブラリである ATLAS をベースにして組み合わせた．そして，最適な適用回数を自動的に選択する機能をもつ行列行列積計算ライブラリを試作し，計算性能の評価を行った．実験の結果，さまざまな行列サイズで ATLAS 単体より高い性能を引き出すことができた．また，通常の方法に比べて誤差がどの程度になるか確認した．

1. はじめに

行列行列積を計算するサブルーチンである DGEMM の性能は，さまざまな科学技術計算をする上で重要である．DGEMM は BLAS[1]の Level-3 に規定されており，DGEMM の性能が向上することで，Level-3 で規定されている他のサブルーチンの性能も向上する．

行列行列積計算を高速化する手法の 1 つに，計算量の削減を行う Strassen のアルゴリズム[2]がある．これは，計算量を$O(N^3)$から$O(N^{\log 7})$まで削減するので，大幅な高速化が期待できる．しかし，実際には行列のサイズがある程度大きくないと通常のアルゴリズムより速くならない．したがって，実用的にはほとんど使われていなかった[3][4].
しかし近年，ハードウェアの性能の向上もあり，大きな行列も扱えるようになってきた．そのため，Strassen のアルゴリズムが有効になる場合が出てきた．それによりさまざまな研究がされている[4][5][6].

Strassen のアルゴリズムは再帰的アルゴリズムであり，N 次正方行列の場合，最大$\log N$回適用できる．しかし，実際には最大回数適用するよりも，適当な回数で通常のアルゴリズムに切り替える方が高速になる場合が多い．Strassen の最適な適用回数は，行列のサイズや計算機の構成によって変わる．したがって，計算機上でよい性能を発揮するには，ハードウェアに合わせてチューニングを行う必要がある．しかし，今日のようにつぎつぎに新しいハードウェアが出てくる状況で，チューニングを行うには多大な手間がかかる．そこで，本研究では Strassen のアルゴリズムを，自動チューニング機能付き線形代数ライブラリである ATLAS[7]をベースに組み合わせた．そして，Strassen の最適な適用回数を自動的に選択する機能を加えた行列行列積ライブラリを試作した．ライブラリの評価実験として，ATLAS 単体との性能比較と計算誤差の確認を行う．

†1 工学院大学
Kogakuin University.

2. Strassen のアルゴリズム

2.1 Strassen のアルゴリズムについて

A,B,C を N 次正方行列とし，それぞれの i 行 j 列の要素をa_{ij},b_{ij},c_{ij}とする．この場合，積の計算$C = A \times B$をするとき，通常のアルゴリズムでは，行列の積の定義式である$c_{ij} = \sum_{k=1}^{N} a_{ik} \times b_{kj}$という式にしたがって計算する．しかし，Strassen のアルゴリズムを使うと，次のように計算することができる．

行列サイズ N が偶数であるとき，それぞれの行列を$N/2 \times N/2$の小行列４つに分ける．

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

このとき，小行列に対し，

$$P_1 = (A_{12} - A_{22})(B_{21} + B_{22}) \qquad P_2 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$P_3 = (A_{11} - A_{21})(B_{11} + B_{12}) \qquad P_4 = (A_{11} + A_{12})B_{22}$$
$$P_5 = A_{11}(B_{12} - B_{22}) \qquad P_6 = A_{22}(B_{21} - B_{11})$$
$$P_7 = (A_{21} + A_{22})B_{11}$$

とおくと，行列 C を次のように求めることができる．

$$C_{11} = P_1 + P_2 - P_4 + P_6 \qquad C_{12} = P_4 + P_5$$
$$C_{21} = P_6 + P_7 \qquad C_{22} = P_2 - P_3 + P_5 - P_7$$

このアルゴリズムでは，$N/2 \times N/2$行列の積の計算が 7 回，和が 18 回である．通常の計算方法では，$N/2 \times N/2$行列の積が 8 回，和が 4 回である．したがって，行列の積の計算量は，通常の方法の$7/8$になっている．一方，Strassen の方法では，行列の和の計算回数が 4 回から 18 回へ増えるため，その分の計算量は増えてしまう．しかし，和のオーダーは$O(N^2)$であるため，N が十分に大きいときは，積の計算量$O(N^3)$に比べて，割合がかなり小さくなる．それにより，積の減少分が和の増加分を上回るため，全体の計算量は減少する．

さらに，小行列のサイズである$N/2$も偶数である場合，7 つの小行列P_1からP_7を同じ方法で計算できるため，アルゴ

リズムを再帰的に用いることができる．つまり，小行列のサイズが偶数であれば Strassen のアルゴリズムを複数回適用することができる. 行列サイズ N が 2 のべき乗であれば，小行列の行列サイズがいつも偶数であるため，小行列が 1×1行列，つまりスカラになるまで用いることができる．それにより，計算量を通常の方法の$O(N^3)$から $O\left(N^{\log 7}\right) \cong O(N^{2.8})$まで減らすことができる．

2.2 N が奇数である場合の適用方法

Strassen のアルゴリズムを用いて行列積計算をする場合は，行列サイズが偶数でなければならない．また，Strassen を複数回適用する場合は，分割した際の小行列のサイズが偶数でなければならない．したがって，一般的な$N \times N$行列に対し，そのまま適用することはできない．

N が奇数であるときに適用する場合は，行列のサイズが偶数になるように，行列をパディングする必要がある．パディングを行うことで，N が奇数でもアルゴリズムを適用することができる．図 1 は N が奇数である行列の例で，図 2 はその行列をパディングしたものである．

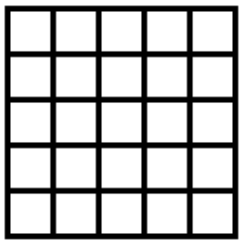

図 1　元の行列

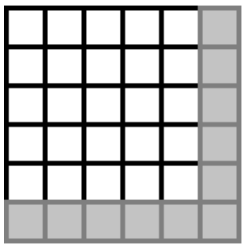

図 2　パディング後

パディングにより，すべての N に対して Strassen を適用できるようになる．また，小行列に対してもパディングすることで，N が 2 のべき乗でない場合でも，アルゴリズムを最大$\log N$回適用することができる．

2.3 Strassen の適用回数と性能

Strassen のアルゴリズムを適用する回数を増やすことで，行列積計算の計算量を減らすことができる．しかし，適用回数を増やし過ぎるとかえって計算量が増えて性能が低下してしまう．例えば，Strassen を 1 回適用する場合，計算量はつぎのようになる．

$$7\left\{2\left(\frac{N}{2}\right)^3-\left(\frac{N}{2}\right)^2\right\}+18\left(\frac{N}{2}\right)^2=14\left(\frac{N}{2}\right)^3+11\left(\frac{N}{2}\right)^2$$

通常の方法の計算量は$2N^3-N^2$であるので，$N \geq 15$でないと計算量が小さくならない．適用する回数を増やすと$O(N^2)$の部分の係数がさらに大きくなるため，より大きい行列サイズでないと計算量は小さくならない．

実際にはこれ以上の N でないと通常の方法に比べて性能は上がらない．要因としては，再帰呼び出しなどにかかるオーバーヘッドの増加がある．また，現在のようなメモリが階層構造になっているハードウェアでは，行列和の計算は積に比べて性能が出にくい．そのため，適用回数を増やして計算上では計算量が減少しても，オーバーヘッドなどによる時間の増加分が，計算量削減による時間の減少分を上回る場合がある．したがって，計算機や行列サイズによって Strassen の最適な適用回数は異なる．よりよい性能を出すためには，適切な適用回数を選ぶ必要がある．

3. BLAS と組み合わせることによる高速化

行列行列積を高速に計算する方法として，BLAS の DGEMM を利用することが挙げられる．この DGEMM に，Strassen のアルゴリズムによる計算量の削減が加わることで，BLAS 単体で計算したときより高速化が期待できる．本研究では，BLAS として，計算機に合わせて自動的にチューニングを行う ATLAS を用いる．

Strassen のアルゴリズムの効果の検証として，ATLAS を単体で使用したものと，ATLAS をベースに Strassen を 1 回から 4 回適用したもので性能を比較する．Xeon X5660 2.80GHz と Intel Core i7 2600 3.4GHz 上で比較した結果を図 3,4 に示す．図中の“ATLAS”は ATLAS を単体で用いたもの，“1 回適用”は Strassen を 1 回適用したものを表す.

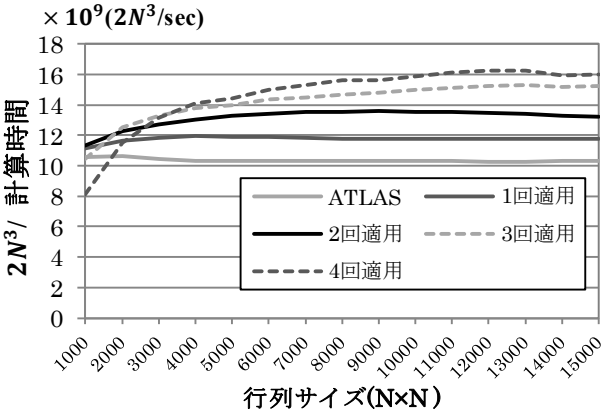

図 3　Xeon X5660 上での比較

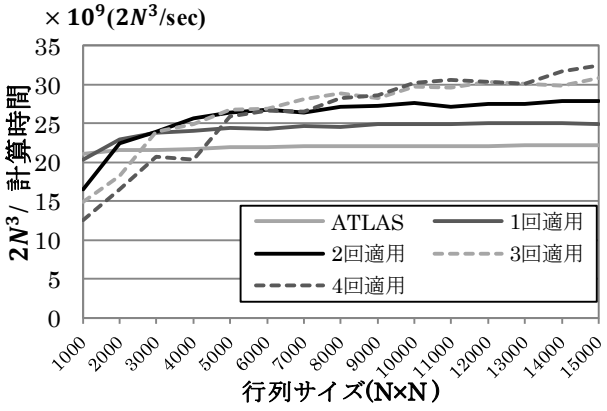

図 4　Intel Core i7 2600 上での比較

また，行列のサイズ N は 1000 間隔で計測している．性能の評価としては，N の 3 乗を 2 倍し計算時間で割ったものを用いる．この値が大きいほど性能が高いことを示す．

結果を見ると，行列サイズ N が小さいうちは，Strassen を組み合わせたものは，Xeon X5660 上では 3 回と 4 回のとき，Intel Core i7 2600 上ではどの回数でも ATLAS 単体より性能は低い．しかし，N が大きくなっていくと計算量削減の効果があらわれ，すべての適用回数で Strassen が ATLAS 単体の性能を大きく上回っている．最終的には，適用回数の多いものがもっとも性能が高くなっている．性能の向上率としては，Xeon X5660 の場合，適用回数が 1 回のときは平均で 14%，4 回のときは 40%向上し，Intel Core i7 2600 では，1 回のとき 10%，4 回で 18%向上した．

4. 行列積自動チューニングライブラリ

4.1 ライブラリについて

本研究では，ATLAS をベースに Strassen のアルゴリズムを組み合わせた自動チューニング機能付き行列積ライブラリを試作した．開発は C 言語で行った．このライブラリは，行列サイズに対して最適な Strassen の適用回数を自動的に選択し，行列積を計算する．

4.2 自動チューニングの必要性

ここでは，試作したライブラリが自動チューニングを行う必要性を示す．行列サイズに対する Strassen の最適な適用回数は，行列サイズや再帰呼び出しのオーバーヘッド，行列和の性能によるため，計算機ごとに変わる．したがって，計算機に合わせて適用回数をチューニングする必要がある．しかし，今日のように非常に多くの計算機がある中で，1 つ 1 つをチューニングをしていくには多大な手間がかかる．そこで，ソフトウェア自動チューニング[8]を行うことで，最適化に要する手間を解消しつつ，高い性能を出すことが期待できる．

4.3 チューニングを行うタイミング

自動チューニングを行うタイミングとしては，インストール時，実行起動前，実行時に分けることができる[8].

Strassen の最適な適用回数は，行列サイズにのみ依存し，行列の要素によって変わることはない．そのため，1 度行列サイズごとの最適な回数が分かれば，次回からはその情報を再利用できるので，再度チューニングを行う必要はない．したがって，チューニングはインストール時に行う．

4.4 ライブラリが行う処理

試作したライブラリでは，インストール時に行う処理のほかに，ユーザプログラム実行時に行う処理がある．それぞれの時点で行う処理内容を図 5 に示す．

まず，ライブラリでは ATLAS を用いているため，ライブラリインストールの前段階として ATLAS の導入がある．ATLAS はインストール時にチューニングを行い，最適化された BLAS を構築する．

ライブラリインストール時では，行列サイズに対する Strassen の適用回数のチューニングを行い，最適なパラメータを記録する．

ユーザプログラム実行時は，ユーザから入力されたデータに対して，最適な Strassen の適用回数を指定して，行列積計算を行う．

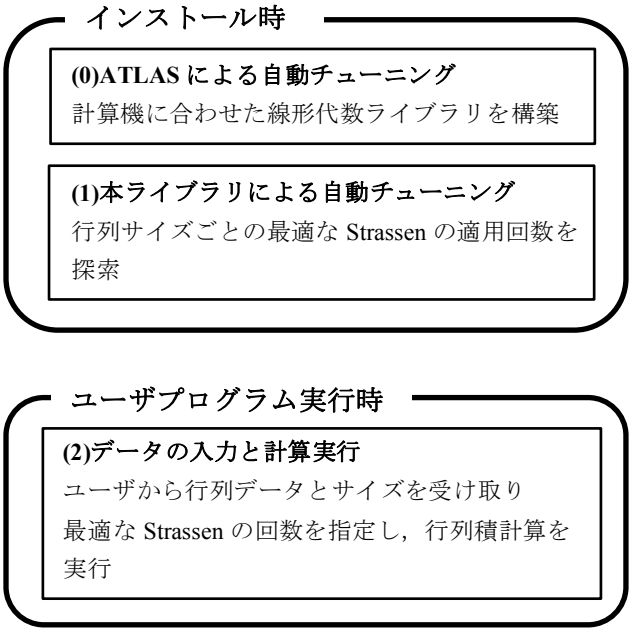

図 5　ライブラリが行う処理内容

4.5 ATLAS と性能周期

ここで，ライブラリにベースとして用いている ATLAS について説明する．ATLAS は，線形代数ライブラリである BLAS の 1 つである．これはインストール時に計算機に合わせてチューニングを行い，最適化された BLAS を構築する．チューニングでは，ループアンローリング段数やブロックサイズの最適化などを行う．これらのパラメータは計算機によって変わる．

ここで，ATLAS の DGEMM の性能には周期があることを示す．

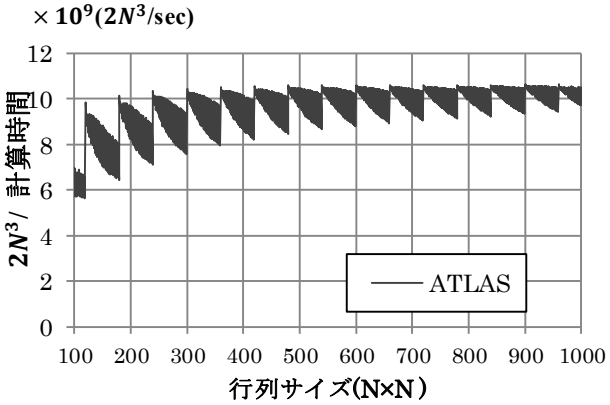

図 6　ATLAS の DGEMM の性能

Xeon X5660 上で，行列サイズ *N*=100 から *N*=1000 の範囲で DGEMM を実行した場合の結果を図 6 に示す．図を見ると，ある *N* で性能が高くなり，そこから徐々に低くなることが等間隔に数回起きていることがわかる．ここで，高い部分から次に高くなるまでを 1 つの周期とする．この周期は計算機によって変わる．Xeon X5660 では性能の周期は 60 である．

4.6 インストール時のチューニング手順

ライブラリインストール時に行うチューニングの手順を図 7 に示す．チューニングは，図 7 に示すように(1-1)から(1-3)の 3 段階の手順がある．これらを順に説明する．

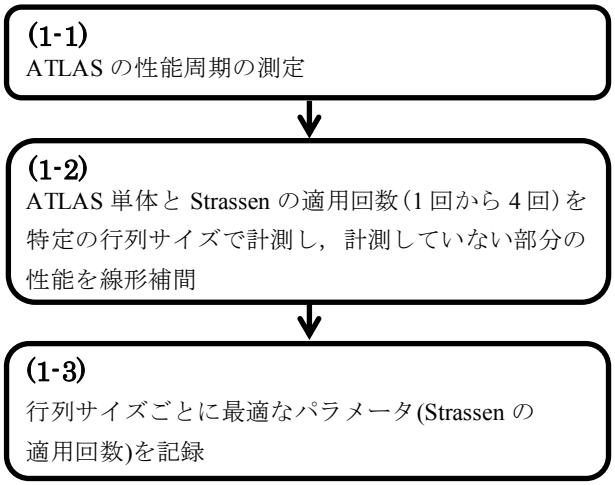

図 7　自動チューニングの手順

第一に，(1-1)の性能周期の測定である．周期を測定する理由は，(1-2)で行う性能計測にかかる時間を削減するためである．**4.5 節**で示したように，ATLAS の DGEMM は性能に周期がある．この周期は，計算機によって異なるため計算機ごとに測定する必要がある．周期を測定するために，まず，行列サイズ *N*=100 から *N*=300 の範囲で ATLAS 単体の性能を計測し，*N* と *N*-1 の計測結果の差をとる．この範囲とした理由は，行列サイズが小さいほうが計測にかかる時間が短いこと，そして，周期の長さは 50 前後であることが多く，最大 200（$= 300 - 100$）あれば十分であると考えたためである．

図 8 は Xeon X5660 上で計測をし，行列サイズ *N* と *N*-1 の計測結果の差をとったものを示している．図 8 を見ると，ひときわ差が大きいところがある．これは，次の周期の最初と，前の周期の最後の行列サイズとの計測結果の差である．周期の中で最初がもっとも性能が高く，最後がもっとも低いため，他のところに比べて差が大きくなる．この性質を利用して ATLAS の性能の周期を求める．

まず，行列サイズ *N* を計測結果の差が大きい順に並べ替える．つぎに，もっとも差が大きくなる *N* と，つぎに大きい *N* の差をとる．その差の絶対値が性能の周期となる．Xeon X5660 の場合では，*N*=120，*N*=180，*N*=240 の順に並べ替わる．したがって，周期は$|120 - 180| = 60$となるため，周期 60 が求まる．

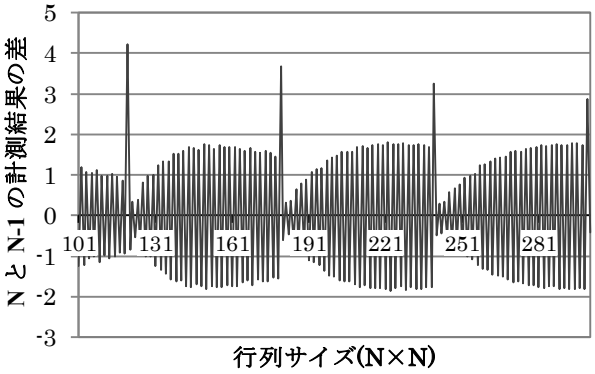

図 8　行列サイズ *N* と *N*-1 における計測結果の差

第二に，(1-2)の性能計測と補間である．ATLAS 単体（適用回数 0 回）と Strassen の適用回数が 1 回から 4 回までの計 5 種類の性能を計測する．今回，ライブラリでは Strassen の適用回数は最大 4 回，計測する行列サイズの範囲は *N*=100 から *N*=4000 までとした．*N*=4000 以降は，Strassen の適用回数が 4 回であるときがもっとも高速になることが多かったため計測しない．行列サイズ *N* は，性能周期の中の最初と最後のみ計測する（ただし，範囲の最小値と最大値である *N*=100 と *N* =4000 も計測する）．これは，性能に周期があることと，周期の最初が高く最後が低いという特徴を利用して，計測する *N* の数を減らしている．計測しない *N* の性能は線形補間する．

また，性能の周期は，Strassen の適用回数が 1 回増えるごとに 2 倍になっていく．したがって，適用回数が増えるたびに計測する *N* は半分に減っていく．

計測する N を周期の最初と最後に限定することで，ATLAS の性能周期を T とすると，計測する N の数は $15/8T(= 2/T \times \sum_0^4 (1/2)^i)$ まで削減される．

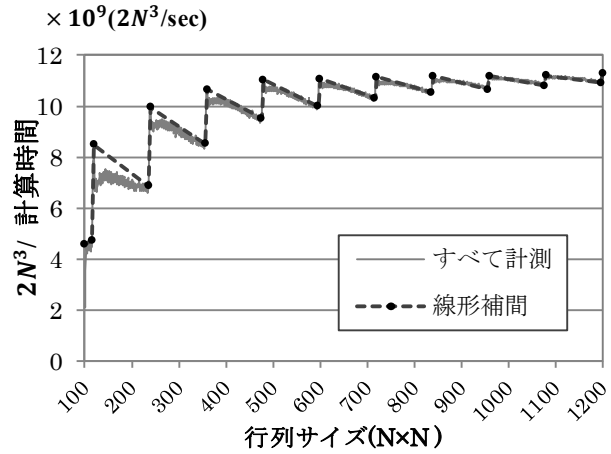

図 9　全計測と線形補間の比較

Xeon X5660 上で，すべての N を計測した場合と線形補間を行った場合の結果を比較した例を図 9 に示す. 図 9 は，Strassen のアルゴリズムの適用回数が 1 回の場合である．この場合，性能の周期は 120（$= 60 \times 2^1$）となる．

第三に，(1-3)の最適パラメータの記録を行う．計測と補間によって得られた性能の結果を図 10 に示す．Strassen の適用回数が増えるごとに計測する N の数が減り，補間されているところが増えていることがわかる. Xeon X5660 では，ATLAS の周期が 60 であるため，計測する N の数は $1/32(= 15/(8 \times 60))$になっている．

この結果を N=100 から N=4000 の範囲で，行列サイズごとに，Strassen の適用回数 0 回から 4 回までの性能を比較し，最適な適用回数を記録する．

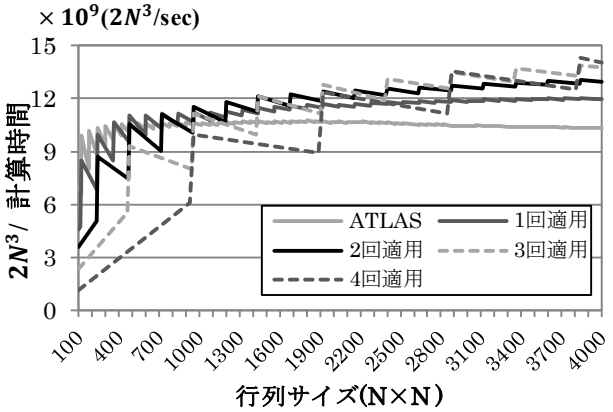

図 10　計測と補間によって得られた性能

4.7 ユーザプログラム実行時の手順

ユーザプログラムがライブラリを呼び出す際のライブラリの処理手順を図 11 に示す．$C = A \times B$という行列積を計算する場合，ユーザは，問題行列データ A,B と行列サイズ N をライブラリに入力する．ライブラリはデータとサイズを受け取ると，それに対する Strassen の適用回数を，チューニングされたパラメータの記録をもとに最適なものを選択し，行列積計算を行う．結果は行列 C へ出力する．

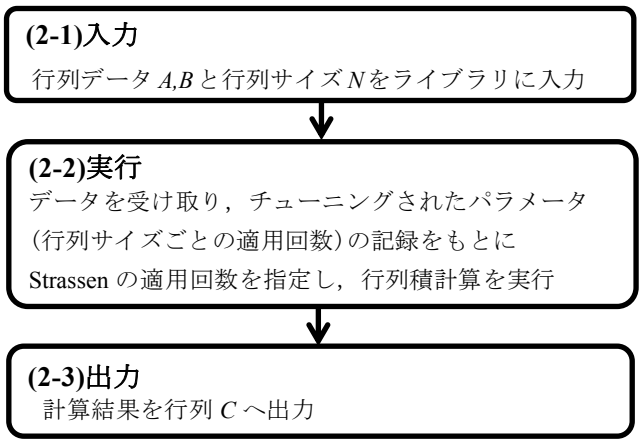

図 11　ライブラリ実行時の処理手順

ここで，試作したライブラリのデータの入力と出力に関しての注意点を説明する．ライブラリには，N が奇数である場合はデータをそのまま入力することはできない．これは， Strassen のアルゴリズムを用いているためである．Strassen は N が偶数でないと適用することができない．したがって，N が奇数である行列は，あらかじめメモリを $(N+1) \times (N+1)$行列として確保しておかなければならない．つまり，N が奇数である場合は，(N+1)行目と(N+1)列目に余分なデータを含んだ結果が出力される．これは，Strassen を 1 度も適用しない，つまり ATLAS 単体として計算される場合も例外ではなく，N は偶数であるとして計算される．したがって，同じ行列サイズでも N が奇数である場合の性能は，ライブラリを使わずに完全に ATLAS のみで計算した場合と異なる．

5. 評価実験

5.1 計算性能の評価

試作したライブラリと ATLAS 単体の行列行列積計算の性能を計測する．実験環境を表 1,2 に示す．計算には，要素が区間[0,1]の一様乱数である倍精度の行列を用いた．

表 1　実行環境 1

CPU	Xeon X5660 2.80GHz
Memory	24GB(DDR-3 SDRAM 1333MHz)
OS	Cent OS 5.3
ATLAS	ATLAS 3.10.0

表 2　実行環境 2

CPU	Intel Core i7 2600 3.4GHz
Memory	32GB(DDR-3 SDRAM 1333MHz)
OS	Cent OS 6.3
ATLAS	ATLAS 3.10.0

行列積計算の性能計測は，チューニングをした行列サイズ N=100 から N=4000 の範囲で行った．実験環境 1,2 においてのライブラリと ATLAS 単体の性能計測の結果をそれぞれ図 12,13 に，ライブラリと ATLAS の性能の比をとったものを図 14,15 に示す．

図 12,13 を見ると，実験環境 1,2 ともライブラリは ATLAS より高い性能が出ている．また，行列サイズが大きくなるにつれて Strassen の適用回数が多いものが選ばれている．

2 つの結果を比較すると，同じ行列サイズであっても Strassen の適用回数が異なっていた．N=1000 から N=2000 の範囲を例にとると，実験環境 1 では主に 2 回と 3 回，実験環境 2 では 0 回と 1 回が選択されていた．.

図 14,15 では，2 つともライブラリの性能が ATLAS 単体より低い部分が見られる．低くなる部分が生じてしまう要因は，チューニング時に計測していない N があるためであ

る．計測していない N の性能は線形補間によって得られたものである．そのため，実際に計測した性能とは異なるので，チューニング時に最適でない Strassen の適用回数が選ばれる可能性がある．それにより，ライブラリの性能が部分的に ATLAS 単体より低くなることがある．

実験環境 1 では，ATLAS 単体に比べて，最大で 2.4%，実験環境 2 では 11%低くなるところがあった．しかし，性能が低くなる部分だけを見ても，実験環境 1 では ATLAS 単体に対して平均 99%，実験環境 2 では平均 98%の性能が出ていたため，問題はないと言える．

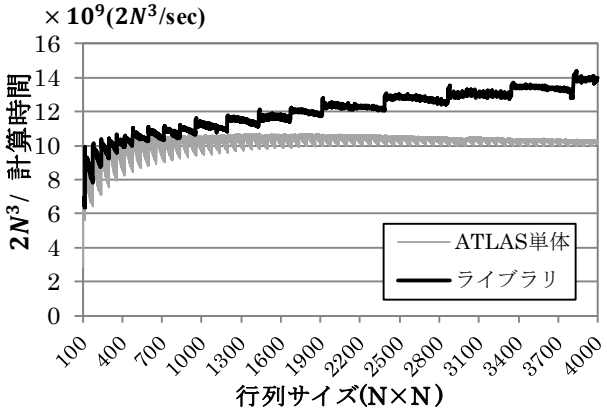

図 12　Xeon X5660 上での性能比較

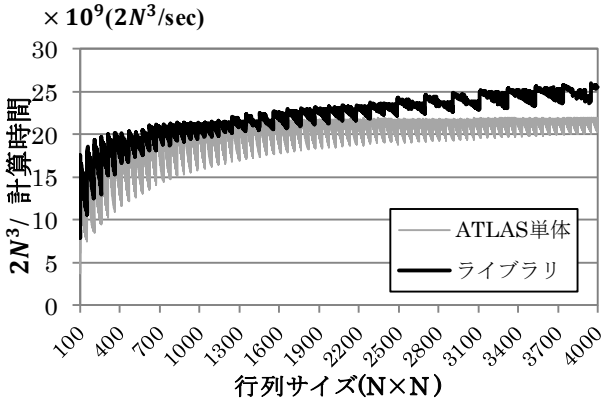

図 13　Core i7 2600 上での性能比較

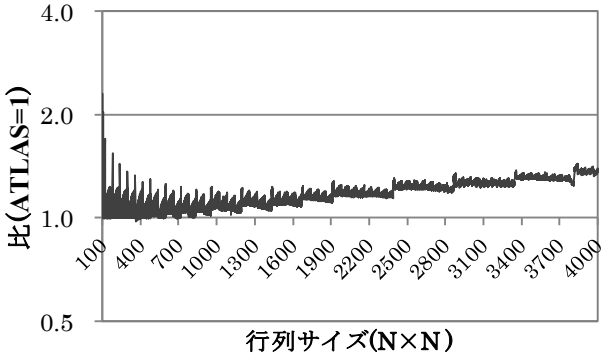

図 14　Xeon X5660 上でのライブラリと ATLAS の性能比

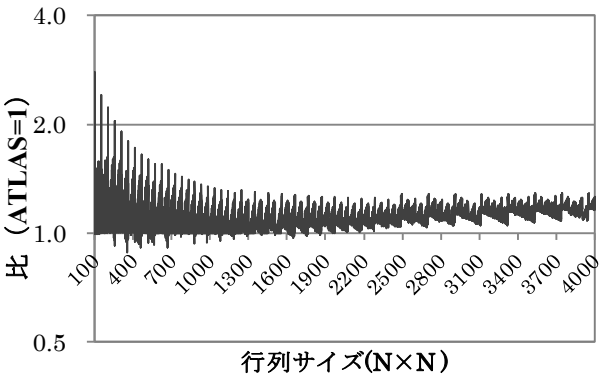

図 15　Core i7 2600 上でのライブラリと ATLAS の性能比

5.2 計算誤差の確認

Strassen のアルゴリズムは，行列の積の定義式にしたがって計算する通常のアルゴリズムに比べ，加減算の回数が増えるために計算誤差が増える可能性がある．そこで，Strassen を適用して計算した場合は，真値に対して誤差がどの程度生じるかを確認した．

確認には，行列の要素が区間(-1,1)の一様乱数である倍精度の行列を用いる．まず，乱数の行列 *A,B* の積を通常の方法と Strassen を適用したものでそれぞれ求める．そして，積のすべての要素について，真値に対する相対誤差を計算する．真値には，行列 *A,B* の要素を 4 倍精度に変換し，通常の方法で計算した結果を用いた．つぎに，積のすべての要素の相対誤差の中で，絶対値が最大のものを最大相対誤差と呼ぶことにする．

誤差の確認では，行列 *A,B* を乱数の種を変えて 2 種類ずつ$(A_1, B_1$と$A_2, B_2)$用意した．それぞれの積$(A_1 \times B_1, A_2 \times B_2)$の最大相対誤差を平均した結果がどれぐらいかを確認する．

行列サイズを変えて計算した結果を表 3 に示す．行列のサイズは 2048,4096,8192 の 3 種類，Strassen の適用回数は最大 4 回までとした．表の“Normal”は，コンパイラによる最適化を行わずに定義式にしたがって計算したときの最大相対誤差で，“ATLAS”は ATLAS 単体での結果である．

表を見ると，行列サイズが大きくなるにつれて，最大相対誤差は増えている．

表 3　2 つの最大相対誤差の平均

回数＼行列サイズ	2048	4096	8192
Normal	3.46×10^{-9}	1.68×10^{-8}	5.58×10^{-7}
ATLAS(0 回)	4.71×10^{-10}	7.87×10^{-9}	5.20×10^{-8}
1 回適用	3.39×10^{-9}	7.27×10^{-9}	2.28×10^{-7}
2 回	3.41×10^{-9}	4.49×10^{-8}	1.84×10^{-7}
3 回	5.15×10^{-9}	4.98×10^{-8}	2.15×10^{-7}
4 回	1.64×10^{-8}	6.13×10^{-8}	1.48×10^{-7}

ATLAS 単体での最大相対誤差は，通常の方法で計算した場合より平均的に小さくなっていた．そして，Strassen を適用した場合は，適用回数が増えると誤差も増えていた．

この方法による確認では，行列サイズが同じである場合，誤差の大きさは 1 桁しか変わらなかった．つまり，Strassen の適用回数が 4 回の場合でも，誤差の増加は 1 桁に収まっていた．

6. おわりに

ATLAS をベースに Strassen のアルゴリズムを組み合わせた自動チューニング機能付きライブラリを試作し，計算性能の評価と計算誤差の確認を行った．結果として，インストール時にチューニングを行った範囲では，ATLAS 単体より高い性能を出すことができた．また，ATLAS より性能が下がる部分についても，問題がないことがわかった．チューニングについては，ATLAS の性質と線形補間を用いることで，計測する行列サイズ N の数を減らして，チューニングにかかる時間を削減した．

計算誤差については，Strassen を適用した場合，相対誤差は最大で 1 桁増加していた．

今後の課題としては，Strassen のアルゴリズムの並列版の実装と評価や，自動チューニングにかかる時間のさらなる削減，パラメータ推定の精度の向上などが挙げられる．特に並列化においては，通常のアルゴリズムより処理の分割が難しく，高い並列化効率を出すのは容易でない．そのため，Strassen のアルゴリズムの並列化がどの程度有効なのか，またどういう処理の分割の仕方がよいかを検証していきたい．

参考文献

[1] BLAS：http://www.netlib.org/blas/.
[2] Volker Strassen：Gaussian Elimination is not Optimal, Numer.Math.，Vol.13，pp.354-356(1969).
[3] 平田富夫：分割統治法，アルゴリズムとデータ構造，pp.137-140(1991).
[4] 竹内純一，萬淳一：Strassen の行列積アルゴリズムの SX-3 上でのインプリメント，情報処理学会第 46 回全国大会講演論文集，平成 5 年前期(1)，pp.57-58(1993).
[5] 後保範：行列乗算におけるストラッセンの方法の拡張，数理解析研究所講究録，pp.61-69(1998).
[6] Paolo D’Alberto, Alexandru Nicolau：Adaptive Strassen and ATLAS's DGEMM: a fast square-matrix multiply for modern high-performance systems，Eighth International Conference on High Performance Computing in Asia Pacific Region，pp.45-52(2005).
[7] Automatically Tuned Linear Algebra Software(ATLAS)：http://math-atlas.sourceforge.net/.
[8] 片桐孝洋，“ソフトウェア自動チューニング”，慧文社，2004.