BLAS と LAPACK を用いたテンソルくりこみ群の数値シミュレーションプログラムの作成

清水　裕也

高エネルギー加速器研究機構　素粒子原子核研究所

1　はじめに

テンソルくりこみ群 [1] は White の密度行列くりこみ群 [2] を元に Levin と Nave によって開発されたスピン系の数値シミュレーション手法である．これらの手法はモンテカルロ法と異なり，符号問題のある系においても有効であるという長所を持つ．しかし，低次元系でしか，高い精度が得られないという弱点も存在する．そのため，モンテカルロ法を置き換えうるものではなく，扱う問題によってそれぞれ使い分けるべきものである．

テンソルくりこみ群はテンソルの分解と縮約の 2 つの操作から成り立っている．テンソルの分解は行列の特異値分解 (SVD) を使って行われ，縮約は行列の掛け算に置き換えられる．これらの行列の線形計算のプログラムは，一から自分で作成しなくとも，BLAS [3] や LAPACK [4] といった数値計算ライブラリを利用することができる．特に，各 CPU に高度に最適化した実装がハードウェアベンダー等から提供されているため，それらを利用することで容易に高い実行性能を得ることができる．

2　テンソルくりこみ群の概要

テンソルくりこみ群はテンソルネットワークを出発点としている．例えば，2 次元系の分配関数のテンソルネットワーク表示は

$$Z = \sum_{i,j,k,l,\cdots=1}^{D} T_{i,m,n,l} T_{s,t,i,j} T_{r,j,k,q} T_{k,l,o,p} \cdots , \tag{1}$$

となる．この式を図にしたものが Fig. 1である．分配関数がスピン変数の数と同数の 4 階のテンソル T によって表されている．一般にテンソルの成分は複素数となる．系が大きな場合には式 (1) を直接計算することはできない．テンソルくりこ

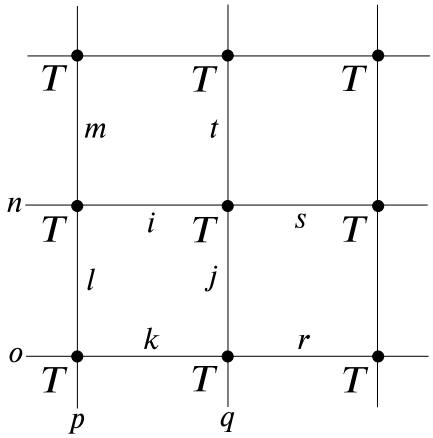

Figure 1: テンソルネットワーク

み群の第一段階として，この 4 階のテンソル T を 2 つの 3 階のテンソルに 2 通りの方法で分解する．

$$T_{i,j,k,l} = \sum_m (S_1)_{j,k,m}(S_3)_{l,i,m}, \tag{2}$$

$$T_{i,j,k,l} = \sum_m (S_2)_{k,l,m}(S_4)_{i,j,m}. \tag{3}$$

そして，次にこれらの 3 階のテンソルを縮約して新しい 4 階のテンソルを作る．

$$T'_{i,j,k,l} = \sum_{m,n,o,p} (S_1)_{m,p,i}(S_2)_{n,m,j}(S_3)_{o,n,k}(S_4)_{p,o,l}. \tag{4}$$

得られた新しいテンソルを使うと，元の半分の数で分配関数を表すことができる．変換毎にテンソルの数は半分になるので，何度も変換を繰り返せば，最終的に分配関数を計算することが可能になる．

3 BLAS によるテンソルの縮約の実装

BLAS を利用したテンソルの縮約部分の実装方法について述べる．まず，テンソルの添字を 2 つずつまとめて次の様な行列を導入する．

$$A_{(ij),(np)} \equiv \sum_m (S_1)_{m,p,i}(S_2)_{n,m,j}, \tag{5}$$

$$B_{(np),(kl)} \equiv \sum_o (S_3)_{o,n,k}(S_4)_{p,o,l}, \tag{6}$$

$$C_{(ij),(kl)} \equiv T'_{i,j,k,l}. \tag{7}$$

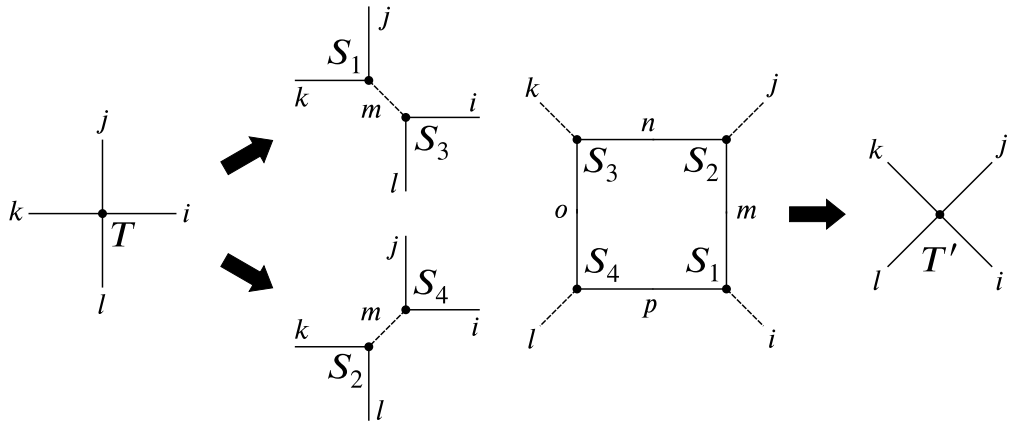

Figure 2: テンソルの分解と縮約

すると，式 (4) は複素行列の積

$$C = AB \tag{8}$$

となる．これは BLAS の Level 3 の zgemm に相当する演算である．行列同士の演算はデータを再利用しやすいので，キャッシュの有効利用が可能であり，うまく実装すれば，高い実行性能が得られる．PC 用の CPU に高度に最適化された BLAS の実装には，CPU ベンダーによるものとして，Intel 社の Intel Math Kernel Library (MKL) や AMD 社の AMD Core Math Library (ACML) がある．また，ソースコードが公開されているものでは，Texas Advanced Computing Center により提供されている GotoBLAS2 [5] や，それを元にして開発が続けられている OpenBLAS [6] がある．これらを利用することで，CPU の理論ピーク性能の 7 割を超える高い実効性能を得ることができる．

4 LAPACK によるテンソルの分解の実装

LAPACK を利用したテンソルの分解部分の実装方法について述べる．式 (2) と式 (3) の分解は，実際には変換によってテンソルの次元が際限なく増大していくのを防ぐため，打ち切り数 D_{cut} を導入して近似的に行われる．

$$T_{i,j,k,l} \simeq \sum_{m=1}^{D_{\mathrm{cut}}} (S_1)_{j,k,m} (S_3)_{l,i,m}, \tag{9}$$

$$T_{i,j,k,l} \simeq \sum_{m=1}^{D_{\mathrm{cut}}} (S_2)_{k,l,m} (S_4)_{i,j,m}. \tag{10}$$

この近似による誤差を D_{cut} の制限のもとでできる限り小さくしなくてはならない．そのために，行列の特異値分解を利用する．まず，テンソルの添字を 2 つずつま

とめて次の様な行列を導入する．

$$M_{(jk),(li)} \equiv T_{i,j,k,l}. \tag{11}$$

この行列の特異値分解を

$$M_{(jk),(li)} = \sum_m U_{(jk),m} \sigma_m V^*_{m,(li)}, \tag{12}$$

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq 0, \tag{13}$$

$$U^\dagger U = V^\dagger V = \mathbf{1}, \tag{14}$$

とする．ここで，$\{\sigma_m\}$ は特異値で，U, V はそれぞれ左右の特異ベクトルを並べた行列である．これらを使って

$$(S_1)_{j,k,m} = U_{(jk),m} \sqrt{\sigma_m}, \tag{15}$$

$$(S_3)_{l,i,m} = V^*_{m,(li)} \sqrt{\sigma_m}, \tag{16}$$

$$m = 1, 2, \cdots, D_{\text{cut}}, \tag{17}$$

と定めると，式 (9) の良い近似が得られる．S_2, S_4 も同様に定める．これは小さな特異値とそれに属する特異ベクトルを捨てることに相当する．小さな特異値は捨ててもその影響は小さい．

LAPACK により提供されている複素行列の特異値分解ルーチンは ZGESVD と ZGESDD の 2 つがある．ZGESVD は QR アルゴリズムに基づいており，数値安定であるが，特異ベクトルの計算量が大きく，大きな行列を扱うのには向かない．ZGESDD は分割統治法を用いており非常に高速である．しかし，必要とする作業領域が大きく，大規模行列を扱うにはより多くのメモリを必要とする．大規模行列に適した高速で必要な作業領域の小さいアルゴリズムとして，MR^3 アルゴリズムがあるが，特異値分解ルーチンはまだ LAPACK に含まれていない．しかし，固有値分解ルーチンを利用した特異値分解が可能である．以下にその方法を述べる．

まず，LAPACK の ZGEBRD を利用して，行列を実上二重対角行列に変換する．

$$B = Q^\dagger M P \tag{18}$$

$$= \begin{pmatrix} a_1 & b_1 & & & \\ & a_2 & b_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & b_{n-1} \\ & & & & a_n \end{pmatrix}. \tag{19}$$

これを使って次の Golub-Kahan 行列を考える．

$$T_{\mathrm{GK}}(B) = \begin{pmatrix} 0 & a_1 & & & & & & \\ a_1 & 0 & b_1 & & & & & \\ & b_1 & 0 & a_2 & & & & \\ & & a_2 & 0 & b_2 & & & \\ & & & b_2 & \ddots & \ddots & & \\ & & & & \ddots & \ddots & b_{n-1} & \\ & & & & & b_{n-1} & \ddots & a_n \\ & & & & & & a_n & 0 \end{pmatrix}. \tag{20}$$

これは実対称三重対角行列なので，MR^3 アルゴリズムを実装した LAPACK の固有値分解ルーチンである DSTEMR を使って固有値と固有ベクトルが求まる．$T_{\mathrm{GK}}(B)$ の固有値には元の行列の特異値とその符号を変えたものが対で現れる．B の左右の特異ベクトルを

$$u = \begin{pmatrix} u_1, & u_2, & \cdots, & u_n \end{pmatrix}^T, \quad v = \begin{pmatrix} v_1, & v_2, & \cdots, & v_n \end{pmatrix}^T, \tag{21}$$

とすると，$T_{\mathrm{GK}}(B)$ の固有ベクトルは

$$q = \frac{1}{\sqrt{2}} \begin{pmatrix} \pm v_1, & u_1, & \pm v_2, & u_2, & \cdots, & \pm v_n, & u_n \end{pmatrix}^T, \tag{22}$$

となっている．B の特異ベクトルを Q と $P^\dagger$ で逆変換すれば元の行列の特異ベクトルが求まる．

DSTEMR は最初に行列が正定値になるようにシフトを行うが，$T_{\mathrm{GK}}(B)$ をシフトした行列から計算した固有ベクトルを使って上記のように特異ベクトルを求めると，小さな特異値に属する特異ベクトルの直交性が著しく悪くなってしまう．これを回避するために，シフトなしの $T_{\mathrm{GK}}(B)$ そのものから MR^3 アルゴリズムを開始する方が良いということが指摘されている [7]．しかし，テンソルくりこみ群においては小さい特異値に属する特異ベクトルは捨ててしまうので，このことは問題にならない．行列の次元が小さい時は ZGESVD や ZGESDD を使い，行列のサイズが大きくなったところで切り替えるようにすれば良い．

5 まとめ

BLAS や LAPACK を用いたテンソルくりこみ群の数値シミュレーションプログラムの作成方法について述べた．テンソルくりこみ群の手続きはすべて BLAS や LAPACK で提供されている行列演算に置き直すことができる．そのため，それらの高度に最適化された実装を利用することで，容易に高い実行性能を得ることができる．

References

[1] M. Levin and C. P. Nave, Phys. Rev. Lett. **99**, 120601 (2007).

[2] S. R. White, Phys. Rev. Lett. **69**, 2863 (1992).

[3] http://www.netlib.org/blas/

[4] http://www.netlib.org/lapack/

[5] http://www.tacc.utexas.edu/tacc-projects/gotoblas2

[6] http://xianyi.github.com/OpenBLAS/

[7] P. R. Willems and B. Lang, ETNA **39**, 1 (2012).