大規模画像データの無転置 2 次元変換法

東京大学生産技術研究所 尾 上 守 夫

ディジタル画像処理に有用なフーリエ、アダマール等の 2 次元変換は、行方向の変換を行い、その中間結果に列方向の変換をほどこしている。画像データ行列が大規模になって補助記憶に行単位で格納しなければならない場合、列方向の読出しは不便なため普通は中間結果を転置しているので演算時間の増加等の不都合が生じる。本論文は転置を要しない新しい演算法を報告する。とくに実数画像の場合は演算時間、記憶容量ともに半分に節減できることを示す。また、たたみ込みおよび相関演算にも本演算法が適合していることを示す。

1. ま え が き

電子計算機によるディジタル画像処理において、フーリエ、アダマールなどの 2 次元直交変換はフィルター、たたみ込み、相関などの演算を行うのに有力な道具である。これらの 2 次元変換は 1 次元変換に分解可能であって、まずデータ行列の各行に 1 次元変換をほどこした後

に列についてさらに変換をほどこせばよい。データ行列がすべて高速記憶に収まる場合は、行方向の変換も列方向の変換も大差はない。しかし一般に画像のデータは膨大であって、高速記憶に収めきれない場合が多い。その場合、データはディスクなどの補助記憶に行単位で順次で格納し、必要に応じて高速記憶内の作業領域に読出して処理し、その結果を再び補助記憶に書込むのが普通である。この際の実出し、書込みも行単位で行われる。したがって、行方向の変換はこの場合でも容易に行える。しかし列方向の変換は列方向の読出しが直接行えないため能率的に行うことができない。

この困難をさけるために、普通は行方向の変換がすんだ時点で、中間結果のデータ行列を転置し、そのうえで再び行方向の変換（等価的に列方向の変換）を行っている。

この方法の欠点は、

- (1) 転置のために巧妙な高速演算法が考えられているが^{2)~4)}、いずれにせよ演算時間が長くなる。
- (2) 最終結果も転置されたままである。
- (3) 出力が入力と同一記憶領域に重ねて書込めるよ

うな高速転置演算法は正方形行列に対してのみ与えられているので、矩形行列の画像に対しては無意味の画素を補って正方形にしたうえで転置する必要がある。

これらの欠点を一掃するために、本論文では、まず転置を要しない新しい 2 次元変換の高速演算法を提案する⁵⁾⁶⁾⁸⁾。

つぎに、2 次元変換として最も広く使用されているフーリエ変換に本演算法を適用する。この場合入力、出力は一般には複素数であってよいが、画像データは長波長ホログラムのような特殊な場合を除き、入力の実数の場合が多い。1 次元フーリエ変換の場合は実数データに対して記憶容量および計算量を半減できるような巧妙な演算法がすでに知られている。本論文はこれを 2 次元に拡張し、無転置演算法でも同様の節減が可能であることを示す。

さらに画像処理でよく使われるたたみ込みおよび相関演算に関しても、1 次元で知られているフーリエ変換を介する高速演算法が、無転置演算法とよく適合して 2 次元に拡張できることを示す⁷⁾⁹⁾。

2. アルゴリズム

1 次元変換の高速演算法は種々あるが、その基本は N ($=R^m$) 点の変換を R 点の変換 N/R 回に分割し、それを m ステップくりかえすことによって演算数を節減しようとするものである。例を最も簡単なアダマール変換の場合にとって説明する。

図 1 は 16 点の 1 次元変換の信号流れ図である。基数 R を 2 に選べば、 m は 4 となって、4 ステップで変換は終了し、その出力を sequence 順に配列替えるために

"A Method for Computing Two-dimensional Transform without Transposing Large Scale Image Data" by Morio Onoe (Institute of Industrial Science, University of Tokyo, Tokyo)

図1 1次元16点アダマール変換の信号流れ図
Signal flow graph for 16 point one-dimensional Hadamard transform.

もう1ステップ必要である。第1ステップでは第0点と第8点をとってその和差をつくり、それぞれもとの場所に書込む。図1で実線は和を、点線は差をとることを示している。つぎに第1および9点に同様にし、順次に演算して第7および15点の和差を求めれば第1ステップは終了する。第2ステップ以降も組になる点の間隔が半減していく以外はまったく同様である。各ステップの演算は直前のステップの出力のみあればよく、しかも同一記憶場所書替型で行うことができる。

以上の特性をふまえて2次元変換は次のように行うことができる。

(1) 第1ステップ ($m=1$)

(1-1) 第0および8行を高速記憶に読出し、それぞれについて行方向の1次元変換を行う。

(1-2) 各行のはじめの点について和差をとり、同一場所に戻す。第2の点……、第16の点についても同様にを行う。結果を補助記憶に書込む。

(1-3) 以上の手続きを第1および9行、……第7および15行までそれぞれ行う。

(2) 第2ステップ ($m=2$)

高速記憶に読出す2行の間隔が第0および4行といったように4行になる他は、第1ステップ(1-2)、(1-3)と同様である。

(3) 第3、第4ステップも行間隔がさらに半減していく他は同様である。

(4) 配列替え

第1行、第15行を読出し、前者を補助記憶の第15行の場所に書込み、さらに第10行を読出してそのあとへ後者を書込むという風に、1行ずつ配列替えを行う。

以上で変換が完了したわけだが、変換のために全データ行列を $m=4$ 回読出し、書込みを行っているわけであ

図2 1次元16点フーリエ変換の信号流れ図
Signal flow graph for 16 point one-dimensional Fourier transform.

る。もし高速記憶の作業領域が4行分 ($R=4$) あれば読出し、書込みを $m=2$ 回にへらすことができる。すなわち第1ステップでは第0, 4, 8, 12行のようにして4行間隔で読出し、行方向の変換を終えたうえで各行の和差を組合せれば図1の $m=2$ のステップまで一挙に進むことができる。同様に第2ステップでは第0, 1, 2, 3行のように1行間隔で読出せば図1の $m=4$ のステップまで進むことができるわけである。

フーリエ変換の場合は各ステップの中間結果に移相因子を掛ける以外はまったく同様である¹⁰⁾。図2に16点の1次元変換の流れ図を示す。ここでは基数 R を4にとっているため、 $m=2$ ステップで変換は終了し、これにディジット逆順の配列替えがもう1ステップ加わる。図中の箱は4点のフーリエ変換を表す。また第1ステップの結果のそばにある数字は掛けるべき移相因子のべき数である。この流れ図から2次元変換はアダマール変換の $R=4$ の場合の読出し、書込みの手続きをそのまま使って遂行できることは明らかである。

4点のフーリエ変換は直接行っても、さらに2点のフーリエ変換に分解しても差支えない。一般に補助記憶と高速記憶との転送は時間がかかるから、高速記憶の余裕があるかぎり R を大きくとって m をへらすのが有利である。またフーリエ変換の単位は2, 4, 8点が便利である。したがって R はその倍数にえらぶのがよい。

その他の2次元変換でも1次元に分解でき、さらにいわゆる高速演算法が存在するものならばこの無転置演算法が適用できることは明らかであろう。

以上の基本アルゴリズムに対して種々の変形が可能である。まず基数は $N=R_1^{m_1} R_2^{m_2} \dots$ のように混合型であっても差支えない。補助記憶に余裕があって結果を同

一場所に収める必要がない場合は、配列替えのステップは省くことができる。第1ステップのはじめに行方向の変換は全部行わずに、それも分解して各ステップに分散させることもできる。これは行方向、列方向の演算がほとんど同じになるので専用のハードウェアを使用する際などに便利であろう。

3. 実数画像データのフーリエ変換

実際の入力画像データの多くは実数である。そのフーリエ変換は複素数になるが、複素共役の対称性をもつ。

図3はそれを視覚的にあらわしたもので、(a)は $N \times N'$ 点の入力画像、(b)はそれに行方向の変換をほどこしたと考えたときの中間結果である。第 $N/2$ 列に関して線対称で、第0列、第 $N/2$ 列は実数になる。(c)はさらに列方向の変換をほどこしたときの最終結果で中心 $(N/2, N'/2)$ に関し点対称で、さらに第0列、第 $N/2$ 列はその中心 $(0, N'/2)$ および $(N/2, N'/2)$ に関し対称である。4点 $(0, 0)$ 、 $(N/2, 0)$ 、 $(0, N'/2)$ および $(N/2, N'/2)$ は実数になる。

冗長な共役数をすて、かつ実数は2個1組に複素数としてパックすれば、中間および最終結果を同一場所に書込むことができる。その配列法はいろいろ考えられるが無転置演算法と組合すには次のようにするのが便利である。

中間結果である図3(b)を $N/2 \times N'$ の複素行列としたものが図3(d)である。実数の第0列、第 $N/2$ 列は

(a) 原データ行列
original data matrix.
(b) 行方向の変換後の中間結果
intermediate data matrix after rowwise transform.
(c) 変換された最終結果
transformed data matrix.
(d) パックした中間結果
packing of intermediate data matrix.
(e) パックした最終結果
packing of transformed data matrix.

図3 2次元フーリエ変換のデータ配列

Data structure of two-dimensional Fourier transform.

パックして第0列に収めてある。これに列方向の変換をほどこすと図3(e)に示すようなやはり $N/2 \times N'$ の複素行列がえられる。この第1〜 $(N/2-1)$ 列は図3(c)のそれと一致しているが、第0列は最終結果の第0および $N/2$ 列の両成分を含んでいる。その分解については後述する。

1次元フーリエ変換の場合、実数データに対して演算を半減するような巧妙な方法がいくつか知られている。そのうち無転置演算法に適しているのは $N \times 2$ 点の実数データを組にして N 点の複素数とみなして変換する方法である。

前章の例について説明すると、実数の第0行、第8行を高速記憶に読出し、それぞれを複素数の実数部、虚数部と見なして行方向の変換を行う。それを分解するわけであるが、これは行方向の配列替えに併せて行うのがよい。すなわち、

$$a_0(i), a_8(i) \quad (1)$$

$$(i=0, 1, \dots, N-1)$$

を原の実数データとする。 B_0, B_8 を図3(d)の形にした対応する中間結果とすれば、次式で与えられる。

$$B_0(0) = R(0) + jI(0) \quad (2)$$

$$B_0(j) = \frac{1}{2}[R(k) + R(k')] + j\frac{1}{2}[I(k) - I(k')] \quad (3)$$

$$B_8(j) = \frac{1}{2}[I(k) + I(k')] + j\frac{1}{2}[R(k') - R(k)] \quad (4)$$

$$(j=1, 2, \dots, N/2-1)$$

ここで、 k, k' は j および $(N-j)$ のビット逆順した数であり、また $[R(i) + jI(i)]$ は $[a_0(i) + ja_8(i)]$ の配列替えをしないビット逆順のままのフーリエ変換である。以下の各行についても同様である。

つぎに図3(e)の第0列の扱いについて考える。この分解は行方向の場合と同様に行うことができるが、列方向の読出しがふくまれるため能率的でない。また同一場所に書込むためには分解した結果を何らかの形でパックする必要が生じる。むしろこのままに残しておいた方が具合がよい。そうしても次節に示すようにフィルター、たたみ込み、相関などの演算に何ら差支えない。

外部に印刷あるいは表示するための出力データは図3(e)に示したように次式で与えられる。

$$C_k(i) + jD_k(i) \quad (5)$$

$$(i=0, 1, \dots, N/2-1, k=0, 1, \dots, N'-1)$$

を計算機内の第 k 行のデータとする。出力の第 l 行のデータは次のようになる。

$$E_l(0) = \frac{1}{2}[C_l(0) + C_{N'-l}(0)]$$

$$+j\frac{1}{2}[D_l(0)-D_{N'-l}(0)] \quad (6)$$

$$\begin{aligned} (j) &= C_l(j) + jD_l(j) \\ \text{for } j &= 1, 2, \dots, N/2-1 \\ &= C_{N'-l}(N-j) + jD_{N'-l}(N-j) \\ \text{for } j &= N/2+1, \dots, N-1 \end{aligned} \quad (7)$$

$$\begin{aligned} E_l(N/2) &= \frac{1}{2}[D_l(0) + D_{N'-l}(0)] \\ &+ j\frac{1}{2}[C_{N'-l}(0) - C_l(0)] \quad (8) \\ (l &= 0, 1, \dots, N'/2) \end{aligned}$$

ここで、離散的フーリエ変換の周期性から次式が成立している。

$$C_{N'}(i) + jD_{N'}(i) = C_0(i) + jD_0(i) \quad (9)$$

最後に逆変換について考える。たたみ込みや相関演算では入力の実数ならば出力も実数である。したがって逆変換への入力は複素共役の対称性を有し、図3(e)の形にパックできる。そこから逆変換は順変換の手続きを一步一步逆にたどればよい。すなわち、まず列方向の変換をほどこすと図3(d)の形の中間結果が得られる。(2)、(3)式を逆に使って N 点の複素数の行をつくり、これに行方向の変換をほどこせば、その実数部、虚数部がそれぞれ対応する実数の2行を与える。

4. たたみ込みと相関

2次元のたたみ込みは次式で定義できる。

$$y(l, m) = \sum_{i=0}^{N-1} \sum_{k=0}^{N'-1} x(i, k) h(l-i, m-k) \quad (10)$$

ここで、 x は図4(a)に示す $N \times N'$ のデータ行列、 h はマスク行列で $Q \times Q'$ の領域で零でない値をもっている。マスクは当初図4(b)に示すように左上隅によせてある。これは中央部に置くやり方¹³⁾にくらべて出力の配列替えが不要で、かつ大規模データの分割と組合せるのに便利である。

X, H をそれぞれ x, h のフーリエ変換とすれば、 y は $(X \cdot H)$ の逆変換として求まる。ただし図4(c)に示すように幅 $(Q'-1)$ および $(Q-1)$ の縁は、まわりこみの誤差によって無意味になる。

つぎに2次元の相関は次式で定義できる。

$$z(l, m) = \sum_{i=0}^{N-1} \sum_{k=0}^{N'-1} x(i, k) h(i-l, k-m) \quad (11)$$

これは $(X \cdot H^*)$ の逆変換で求まる。ただし*は共役複素数である。図4(d)に示すように今度は下側および右側の縁が無意味になる。

x, h が実数ならば X, H は前節の方法で求まり、図3(e)の形に収められる。したがって、その積は両者の

(a) 原データ行列
original data matrix.
(b) マスク行列
mask matrix.
(c) たたみ込み演算の結果
result of convolution.
(d) 相関演算の結果
result of correlation.

図4 2次元たたみ込みおよび相関演算のデータ配列
Data structure of two-dimensional convolution and correlation.

対応する行を読出せば容易に作ることができる。ただし第0列にあたる第1要素は前節に述べたパックのため特別な取扱いが必要である。すなわち、

$$\begin{aligned} X \cdot H \text{ or } X \cdot H^*(0, k) \\ = \frac{1}{4}(1-j)(F \cdot G + F'^* \cdot G'^*) \\ + \frac{1}{4}(1+j)(F \cdot G'^* + F'^* \cdot G) \end{aligned} \quad (12)$$

ここで、

$$\begin{aligned} X(0, k) &= F, H \text{ or } H^*(0, k) = G \\ X(0, N'-k) &= F', H \text{ or } H^*(0, N'-k) = G' \end{aligned} \quad (13)$$

たたみ込みおよび相関では、さらに次のような演算の節減が可能である。

- (1) 行方向、列方向の変換とも配列替えは省いてよい。
- (2) 列方向の逆変換の最終ステップにおいて、図4(c)の上の縁あるいは図4(d)の下縁に相当する行は捨ててしまってもよい。したがって、その後の行方向変換も不要になる。

以上の2項は一般の複素数の場合およびデータ全部が高速記憶に収まる場合でも有効である。

データ行列の行方向の長さが非常に大きくなった場合、たたみ込みおよび相関は分割して行うことが必要になる。1次元の場合は overlap-save 法¹⁴⁾と overlap-add 法¹⁵⁾と呼ばれる巧妙な演算法がある。前者はデータを互いに $(Q-1)$ 点ずつオーバーラップした N 点の区間に分割し、それぞれに Q 点のマスクをたたみ込む。その結

果の各区間の最初の $(Q-1)$ 点を捨てて残りをつなぎ合
せれば全体のたたみ込みが得られる。

後者はデータを $(N-Q+1)$ 点の区間に分割し、各区
間の末尾に $(Q-1)$ 点の零を補って N 点の長さにする。
それぞれに Q 点のマスクをたたみ込み、その結果を $(Q-1)$ 点ずつオーバーラップさせてつなぎ合すると全体のた
たみ込みが得られる。オーバーラップしたところは和をと
るものとする。

相関の場合も両者に対応した分割法が存在する。

以上を2次元に拡張する際は overlap-save 法の方が
中間結果をシフトするだけで最終結果が得られるので便
利である。補助記憶が余裕ある場合にはシフトすら不要
であって、そのままにしておけばよい。出力のときに無
意味な縁を捨てればよいからである。

図5は2次元の原データ行列の分割法を示したもので
ある。

第 $[p, q]$ 区間の左上隅の点は次式で与えられる。

$$p \cdot (N-Q+1), q \cdot (N'-Q'+1) \quad (14)$$

もちろん分割数は少ない方がよい。無転置演算法は少
なくとも2行を高速記憶に読出す必要がある。したがっ
て原行列の行方向の長さが作業領域の半分を越す場合に
は行方向の分割が必要である。これに対して列方向の長
さは制限がないから分割する必要がない。転置した場合
には行方向、列方向ともに分割が必要になるので、この
点も無転置演算法がすぐれている。

データ行列が大規模の場合、マスク行列は大部分が零
になるので、次のような演算の節減が可能である。まず
1次元の場合を考え、 N 点中はじめの Q 点だけが零で
ないとする。末尾に適当に零を補って、あるいは Q に
若干の零を含むようにして N が Q の倍数になるよう
にする。

図5 2次元 overlap-save 分割法
Two-dimensional overlap-save sectioning

$$N=L \cdot Q \quad (15)$$

x のフーリエ変換 X は次式で定義される。

$$X_n = \sum_{k=0}^{N-1} x_k W_N^{kn} \quad (16)$$

ここで、

$$W_N = \exp\left(-j \frac{2\pi}{N}\right) \quad (17)$$

$$x_k \neq 0 \text{ for } k=0, 1, \dots, Q-1$$

$$= 0 \text{ for } k=Q, Q+1, \dots, N-1 \quad (18)$$

n を L を単位として次のように表すと、(16)式は次の
ように書き直せる。

$$n = n_1 L + n_0,$$

$$n_1 = 0, 1, \dots, Q-1, n_0 = 0, 1, \dots, L-1 \quad (19)$$

$$\begin{aligned} X_n &= \sum_{k=0}^{Q-1} x_k W_N^{kn_0} W_N^{kn_1 L} \\ &= \sum_{k=0}^{Q-1} [x_k W_N^{kn_0}] W_Q^{kn_1} \end{aligned} \quad (30)$$

ここで、 $[\]$ の内は n_1 に関係しない。したがって、
 X_n 全体は Q 点の変換 L 回で求まる。したがって演算
の節減比は次式で与えられる。

$$\frac{L \cdot Q \ln Q}{N \ln N} = \frac{\ln Q}{\ln N} \quad (21)$$

以上の手順を2次元に拡張するのは容易であって、改
めて説明を要しないであろう。

なお、本節に述べた分割法は一般の複素数の場合にも
適用できる。

5. む す び

大規模な画像データ行列を行単位で補助記憶に収めた
場合に、中間結果を転置しないで2次元変換を行う新し
い演算法を考案した。とくに2次元フーリエ変換の場合
をくわしく検討し、入力を実数の場合は演算時間、記憶
容量ともに半分に節減できることを示した。また、たた
み込みおよび相関演算の場合も無転置演算法がうまく適
用できることを示した。

この演算法は当所の計算機に対してプログラムされて
便利に使用されている。その詳細は別に報告する予定で
あるが、参考のため演算時間に関する資料を付録にまと
めておいた。

(昭和50年4月17日受付)

〔参 考 文 献〕

- 1) B. R. Hunt: Data structures and computational organization in digital image enhancement, Proc. IEEE, 60, July (1972) 884-887
- 2) J. O. Eklundh: A fast computer method for matrix transposing, IEEE Trans., C-21, July. (1972) 801-803

- 3) U. Schumann: Comments on "A fast computer method for matrix transposing" and application to the solution of Poisson's equation, IEEE Trans., C-22, May (1973) 542-543
- 4) 高木, 横井: 転置行列を高速に得る方法について, 信学会電子計算機研資, EC 73-26 (1973.9)
- 5) 尾上: 大規模2次元FFTの高速演算法, テレビ大全, 11-1 (1974)
- 6) 尾上: 行列の転置を要しない大規模2次元変換の高速演算法, 東大生研共同研究「多次元情報の伝送と処理」資料 (1974.6.12)
- 7) 尾上: 2次元FFTの無転置演算法の実数データ行列への適用, 同上 (1974.10.23)
- 8) M. Onoe: A method for computing large-scale two-dimensional transform without transposing data matrix, Proc. IEEE., 63, Jan. (1975) 196-197
- 9) 尾上: 2次元DFTの無転置演算法, 昭50信学大全, S 5-12
- 10) W. M. Gentleman and G. Sande: Fast Fourier transforms-for fun and profit, 1966 Fall Joint Comput. Conf., AFIPS Proc., 29 (1966) 563-578
- 11) G. D. Bergland: A fast fourier transform algorithm for real valued series, Comm. ACM., 11, Oct. (1968) 703-710
- 12) 高橋: 高速フーリエ変換について, 情報処理, 14, 8 (1973) 616-622
- 13) P. E. Anuta: Spatial registration of multispectral and multitemporal digital imagery using fast fourier transform techniques, IEEE, Trans., GP-8, 4, Oct. (1970) 353-368
- 14) T. G. Stockham: High speed convolution and correlation with application to digital filtering, Chapter 7 in "Digital Processing of Signals" by B. Gold and C. M. Rader (McGraw Hill, N. Y. 1969)
- 15) T. G. Stockham: High speed convolution and correlation, Spring Joint Comp. Conf., AFIPS Proc., 28, (1966) 229-233

〈付録〉 2次元フーリエ変換の演算時間

使用した計算機は16ビット, 32k語のミニコンピュータ(YHP 2100 A)で, 補助記憶は磁気ディスクを用いた。プログラムはディスクオペレーティングシステムの管理下にあるフォートランで書いた。コア内の作業領域は16k語(4k複素語)にとって, フーリエ変換の基数は4である。

付・表1に計時結果を示す。細分した時間は大体の目安であるので, 総和は, 必ずしも総合時間と一致していない。

付・表1 2次元フーリエ変換の処理時間(単位: 秒)
Processing time of two-dimensional Fourier transform by the new algorithm (unit: sec)

	64×64		256×256		
	複素数	実数	複素数	実数	整数
処 理	20.5	—	416	272	172
ディスク I/O	なし	なし	288	144	20
配列替え	3	—	187	73	68
総 合	25.1	9.2	895	479	261
マイクロプログラム	10	—	340	—	—

64×64の演算はすべてコア内で行われているので, これを基準にして256×256に無転置演算法を適用した場合の効率が見られる。2次元FFTの演算回数は $N^2 \log N^2$ に比例すると考えると, 後者は前者の21.3倍すなわち437秒になるはずであるが, 実測の416秒はこれに近い。したがって無転置演算法が無駄な演算をやっていないことがわかる。転置をやればその時間がこれに加算されることになる。ディスクの入出力時間には待合せ時間がかかり入っていて, これは多重プログラムの大型計算機なら他に有効に使えるはずのものである。配列替えは, 列方向の時間のみ示してある。ディスクに余裕があれば, これは省ける。

実数データに対しては予想どおり演算時間がほぼ半減している。複素数および実数は浮動小数点を用いたが, これを固定小数点の整数にしたものも参考のため最右欄に示してある。

最下行は一部にマイクロプログラミングを用い, またディスクの入出力をオペレーティングシステムの管理を省略した形で行った場合で, かなり演算時間が短縮されている。

なお, 以上のプログラムは大学院学生稲本康君, マイクロプログラムは同じく岩下正雄君の努力に負うところが多い。

〔付録参考文献〕

- A1) 尾上, 稲本: 2次元FFTの処理時間の比較, 東大生研共同研究「多次元情報の伝送と処理」資料 (1975.2.27)