ア ル ゴ リ ズ ム 1－3
（１９８８. 5. ２３）

ドットパターンを回転するアルゴリズム

中森眞理雄
東京農工大学工学部数理情報工学科

１語がｎビットの計算機において、連続するｎ語にｎ×ｎドットパターンがあるとする。すなわち、このパターンの第ｉ行第ｊ列にドットがあるならば第ｉ語の第ｊビットの値は１で、さもなければその値は０とする（ｉ，ｊ＝０，１，…，ｎ－１）。このパターンを９０°回転する問題、すなわち、第ｉ語の第ｊビットを第（ｎ－１－ｊ）語の第ｉビットに移す問題を考える。本論文では、この問題に対して手間が０（ｎ log ｎ）のアルゴリズムを提案する。このアルゴリズムを少々手直しするだけで１８０°、２７０°回転するアルゴリズムも導かれる。ドットパターンを回転するアルゴリズムと高速フーリエ変換との類似点についても述べる。

An O(n log n) Algorithm of Rotating an n×n Bit Pattern in a Computer

Mario Nakamori
Department of Information Science
Tokyo University of Agriculture and Technology
2-24-16, Nakamachi, Koganei, Tokyo 184, Japan

Suppose a dot pattern (figure, letter, symbol, etc.) is represented as a sequence of n words of a computer memory, where each word consists of n bits. If a dot exists in the i-th row and the j-th column (i, j=0, 1,..., n-1) of the matrix, then the j-th bit of the i-th word is 1; otherwise, the bit is 0. Then our problem is as follows: Rotate counterclockwise by 90° the bits in these n words, i.e., move the j-th bit of the i-th word to the i-th bit of the (n-1-j)-th word. In the present paper we propose an O(n log n) algorithm for this problem. Also, the analogy between this algorithm and the fast Fourier transform is considered.

1．はじめに

文字，記号，図形などのパターンがｎ行ｎ列のドットマトリックスとして計算機の記憶装置の中に格納されているとする．ここにｎはこの計算機の１語の長さとし，このパターンはこの計算機の連続するｎ語を占めるものとする．このパターンの第ｉ行第ｊ列にドットがあるならば第ｉ語の第ｊビットの値は１で，さもなければその値は０とする（ｉ，ｊ＝０，１，…，ｎ－１）．ただし，語のもっとも左のビットが第０ビットで，もっとも右のビットが第ｎ－１ビットとする．このとき，このパターンを９０°回転する問題，すなわち，第ｉ語の第ｊビットを第（ｎ－１－ｊ）語の第ｉビットに移す問題を考える（図１）．

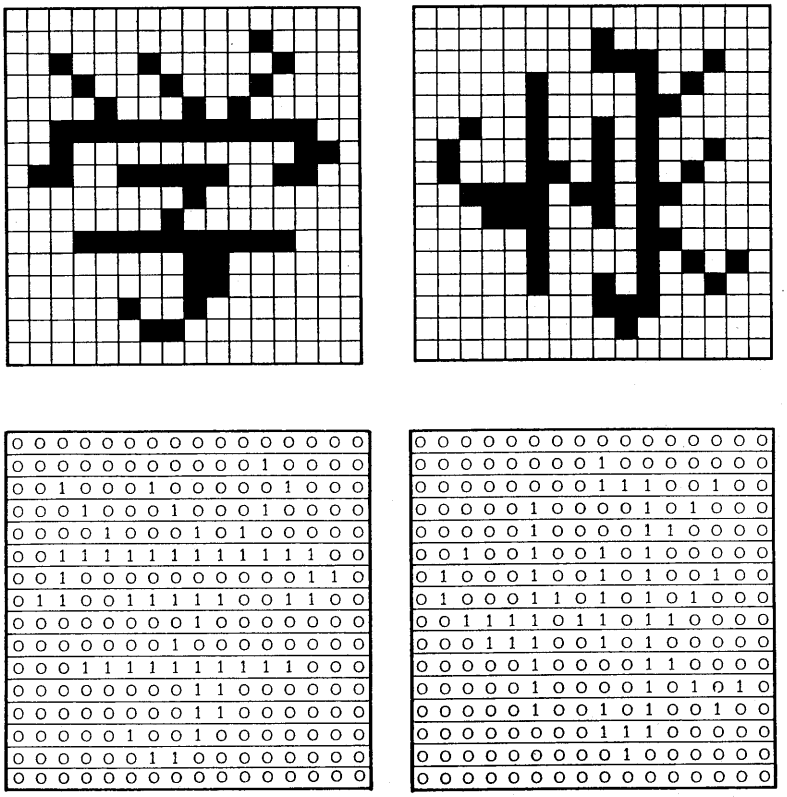

図１　ドットパターンの回転

この問題はプリンタやディスプレイの設計などに現れることがある．

同じパターンを９０°，１８０°，２７０°回転したり，上下，左右に反転したりすることはしばしば必要となることである（例えば，将棋や麻雀の記録）．回転されたり反転されたりしたパターンを別のパターンとして記録しておくことも考えられるが，それには一つのパターンを記録するのに比べて何倍もの記憶場所が必要である．一つのパターンだけを記録しておいて，使う際にはそのコードと回転・反転の情報とを併せて用いる方法はデータ通信の量を倍にし，回転・反転の手間を要する．どちらが良いかは状況によるが，それはともかく，回転・反転の高速なアルゴリズムを開発しておくことは意義のあることであろう．

素朴にｎ^{2}個のビットの各々を対応する場所に移動すれば回転・反転の目的は達せられる．しかし，それでは$O(n^2)$の手間を要する．ｎ×ｎドットパターンを上下・左右の４つに分割することを繰り返して４分木（quadtree）として表現すれば，回転や反転はポインタのつけ換えでできるが，データを４分木に変換するために$O(n^2)$の手間を要する．本論文では，ｎ×ｎドットパターンを$O(n \log n)$の手間で回転したり反転したりするアルゴリズムを提案する．

本論文の動機は，日本語浄書システムにおけるプリンタ部の設計に由来している．そこでは，ＣＰＵにＭ

C68000を用い，一つの32×32ドットパターンを5ミリ秒以内で回転したり反転したりすることが要求された．素朴なアルゴリズムではビットの移動を1024回繰り返すから，1回のビットの移動を約5マイクロ秒以内で行わなければならない．もっとも単純なAND命令にすら1マイクロ秒を要するMC68000にとって，この条件を満たすことは困難である．より高速のCPUを使うことも考えられるが，その場合は処理速度に対する要求も同時に厳しくなるはずであり，根本的な解決にはならない．本論文で提案するアルゴリズムは一つの32×32ドットパターンを約3ミリ秒で回転したり反転したりすることができ，十分に目的は達せられる．

本論文のアルゴリズムは，部分パターンの回転を複数並行して行うという考えに基づいている．この考えは高速フーリエ変換にも見られるものである．本論文では，ドットパターンの高速回転と高速フーリエ変換との類似点についても述べる．

2．90° 回転アルゴリズムR_{90}

本アルゴリズムはn×nドットパターンを同じ大きさの部分パターンに分割し，横に並んだ部分パターン全部の回転を同時に行う．図2では16個のブロック（部分パターン）に分割し，横に並んだ4個のブロック（たとえば，図の“A”，“B”，“C”，“D”）を同時に回転する．

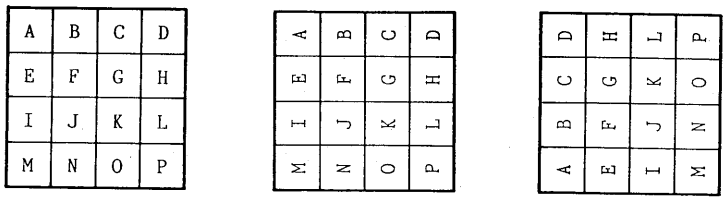

図2　部分回転

アルゴリズムを詳細に説明するためには，“小ブロックを単位とした回転”と“中ブロック内の回転”の概念が必要である．nは素数ではないとして，n＝kℓmとする．n×nドットパターンを大きさk×kのℓ^2m^2個のブロックに分割する．大きさkℓ×kℓのm^2個のブロックがあると解釈することもできる． 前者を小ブロック，後者を中ブロックとするとき，次数（k，ℓ）の回転とは大きさk×kの小ブロックを単位として大きさkℓ×kℓの中ブロック内で回転することとする．正確には，次数（k，ℓ）の部分回転とは，もとのパターンにおける第（kℓu＋kv＋w）語の第（kℓp＋kq＋r）ビットを新しいパターンにおける第（kℓu＋k（ℓ－1－q）＋w）語の第（kℓp＋kv＋r）ビットに移すことである（ただし，p，u＝0，…，m－1； q，v＝0，…，ℓ－1；r，w＝0，…，k－1）（図3）．次数（k，ℓ）の回転は次の手続き subrotate(source,destination,k,ℓ,m,mask) によって実行される．ただし，sourceとdestinationはもとのパターンと回転後のパターンの存在する領域である．

```
procedure subrotate(source, destination, k, l, m, mask);
  array source, destination, mask;
  integer k, l, m, p, q, r, v;
    for p := 0 step 1 until m − 1 do
      for r := 0 step 1 until k − 1 do
        for v := 0 step 1 until l − 1 do
          begin
            for q := 0 step 1 until l − 1 do
              begin
                register[q] := source[k × l × p + k × q + r];
                register[q] := register[q] ∧ mask[v];
                shiftright(q, k × (q − v))
              end;
            for q := 1 step 1 until l − 1 do
              register[0] := register[0] ∨ register[q];
            destination[k × l × p + k × q + r] := register[0]
          end
```

上記の手続きにおいて∧と∨はそれぞれ語のビットごとのAND演算とOR演算である．mask はℓ語の配列で，第

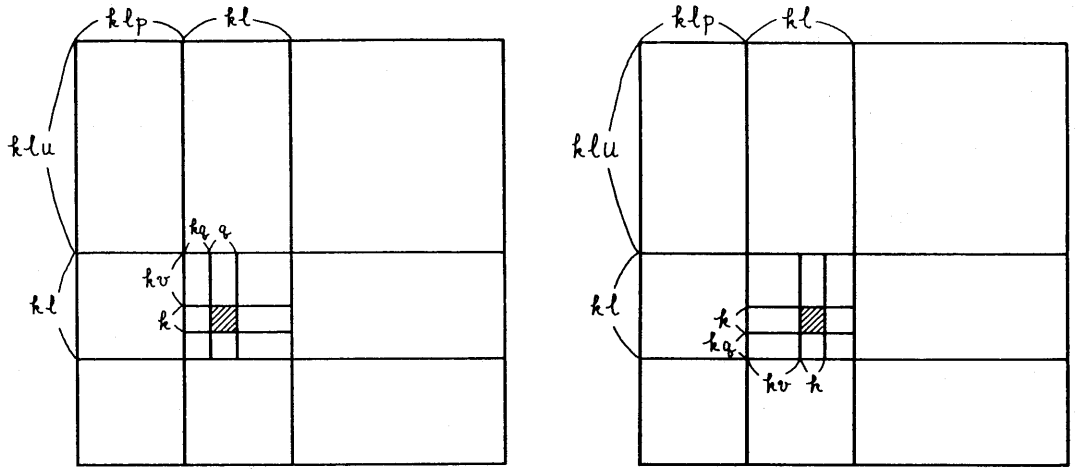

図3　次数（k，ℓ）の部分回転

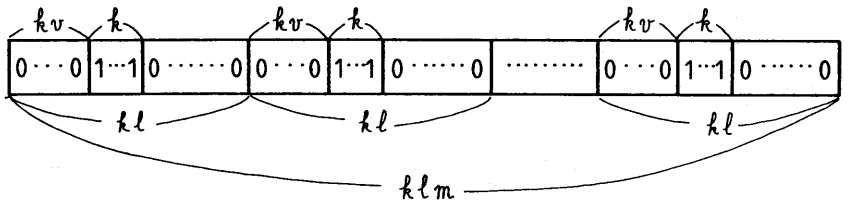

図4　ｍａｓｋ［ｖ］

ｖ要素 mask［ｖ］（ｖ＝０，１，…，ℓ－１）の第ｊビットは

あるｕ（＝０，１，…，ｍ－１）に対して　ｋℓｕ＋ｋｖ≦ｊ＜ｋℓｕ＋ｋ（ｖ＋１）のとき　　１；

上記以外のとき　０，

とする．mask［ｖ］は符号なし整数

$$2^{k(\ell-v-1)}\sum_{w=0}^{k-1}2^{w}\sum_{u=0}^{m-1}2^{k\ell u}=2^{k(\ell-1-v)}(2^{k}-1)(2^{n}-1)/(2^{k\ell}-1)$$

と解釈することもできる．手続 shiftright(i,j) は register[i] のビットをｊビット右にシフトする．ただし，ｊ＝０なら register[i] の内容は変わらず，ｊ＜０なら register[i] のビットは－ｊビット左にシフトする．

mask［ｖ］が符号なし整数2^{n-v-1}，すなわち，

$$\overbrace{0\cdots0}^{v}\,1\,\overbrace{0\cdots0}^{n-v-1}$$

のときは，subrotate (source, destination, 1, n, 1, mask)
は素朴なＯ（n^2）のアルゴリズムと変わりがない．また，mask［ｖ］が符号なし整数2^{n-1}，すなわち，

$$\overbrace{1\cdots1}^{n}$$

のときは，subrotate (source, destination, n, 1, 1, mask) は回転しない（すなわち恒等変換）のに等しい（ただし，手間はＯ（ｎ）であり，何もしないわけではない）．

ｎ×ｎドットパターンを９０°回転するアルゴリズムR_{90}は，上記の subrotate を用いて次のように記述される．

まず，$n=k_1k_2\cdots k_t$と分解する．その上で，次のステップ１，ステップ２，…，ステップｔを実行する．ただし，work2, work3, …, worktはｎ語の連続する作業場所で，source, work2, work3, …, workt, destination は互いに重ならないものとする．

ステップ１　　subrotate (source, work2, 1, k_1, $k_2\cdots k_t$, mask1)

ただし，MASK1は大きさk_1の配列で，mask1[v] (v=0,…,k_t-1)の値は$2^{k_1-1-v}(2^1-1)(2^n-1)/(2^{k_1}-1)$

ステップ２　subrotate (work2, work3, k_1, k_2, $k_3 \cdots k_t$, mask2)

ただし，MASK2は大きさk_2の配列で，mask2[v] (v=0,…,k_2-1)の値は$2^{k_1(k_2-1-v}(2^{k_1}-1)(2^n-1)/(2^{k_1 k_2}-1)$

⋮

ステップｔ　subrotate (workt, destination, $k_1 \cdots k_{t-1}$, k_t, 1, maskt)

ただし，MASKtは大きさk_tの配列で，maskt[v] (v=0,…,k_t-1)の値は$2^{k_1 k_2 \cdots k_{t-1}(k_t-1-v}(2^{k_1 k_2 \cdots k_{t-1}}-1)(2^n-1)/(2^{k_1 k_2 \cdots k_t}-1)$

３．R_{90}の手間

手続き subrotate(source,destination,k,ℓ,m,mask) は

ＭＯＶＥ命令	ｋℓ^{2}ｍ＋ｎ 回
ＡＮＤ命令	ｋℓ^{2}ｍ 回
ＯＲ命令	ｋℓ（ℓ－１）ｍ 回
ＳＨＩＦＴ命令	ｋℓ^{2}ｍ 回

から成るので，手間はＯ（ℓｎ）である（ｋℓｍ＝ｎであることに注意）．したがって，ステップ１，ステップ２，…，ステップｔの手間はそれぞれＯ（k_1ｎ），Ｏ（k_2ｎ），…，Ｏ（k_tｎ）である．これらを合わせると，R_{90}の手間は全体でＯ（（$k_1+k_2+\cdots+k_t$）ｎ）である．なお，ここではｎを積$k_1 k_2 \cdots k_t$に分解する手間は考えていない．また，mask1，mask2，…，maskt は immediate data としてプログラム中に定数として直接埋め込むことができるので，それらを作り出す手間も考えていない．

ｎ＝2^t のときは，含まれる命令の数は，ＭＯＶＥ命令 ３ｎ $\log_2$ ｎ 回，ＡＮＤ命令 ２ｎ $\log_2$ ｎ 回，ＯＲ命令 ｎ $\log_2$ ｎ 回，ＳＨＩＦＴ命令 ２ｎ $\log_2$ ｎ 回であり，手間はＯ（ｎ log ｎ）である．

冒頭に述べたプリンタでは，ｎ＝３２であるから，ＭＯＶＥ命令 ４８０回，ＡＮＤ命令 ３２０回，ＯＲ命令１６０回，ＳＨＩＦＴ命令 ３２０ 回である．そこでは，アセンブリ言語を用い，direct あるいは immediate アドレッシングモードを用いているので，for ループやアドレス計算などの手間は無視できる．ＭＣ６８０００では，それぞれの命令の実行時間は，ＭＯＶＥ命令 １９ クロック，ＡＮＤ命令 ７ クロック，ＯＲ命令 ２５ クロック，ＳＨＩＦＴ命令 ９＋２ｓ クロック（ｓはシフトするビットの数，ここではｓ＝１，２，４，８，１６が同じ回数ずつ）である．１クロックは１２５ナノ秒なので，計算時間は３．２８１ミリ秒となり，５ミリ秒以下という当初の目標は達成された．

もし，ｎ＝３２ のとき subrotate (source, destination, 1, 32, 1, mask) （ただし，mask［ｖ］は符号なし整数2^{n-v-1}）によって回転するのであれば，それぞれの命令の実行回数は１０２４ 回であるので，全体の実行時間は７．９３６ミリ秒となり，５ミリ秒以下という当初の目標は達せられない．

４．アルゴリズムR_{180}，R_{270}

１８０° の回転はR_{90}を２回行えばよく，手間もオーダーとしてはR_{90}と同じであるが，ｎ＝３２のとき５ミリ秒以下で行うという目標は達せられない．そこで，R_{90}とは別のアルゴリズムを考えることにする（ただし，考え方はR_{90}と同じである）．

まず，次数（ｋ，ℓ）の部分反転を定義する．もとのパターンにおける第（ｋℓｕ＋ｋｖ＋ｗ）語の第（ｋℓｐ＋ｋｑ＋ｒ）ビットを新しいパターンにおける第（ｋℓｕ＋ｋ（ℓ－１－ｖ）＋ｗ）語の第（ｋℓｐ＋ｋ（ℓ－１－ｑ）＋ｒ）ビットに移すことである（ただし，ｐ，ｕ＝０，…，ｍ－１； ｑ，ｖ＝０，…，ℓ－１；ｒ，ｗ＝０，…，ｋ－１）．次数（ｋ，ℓ）の部分反転は次の手続き subtranspose(source,destination,k,ℓ,m,mask) によって実行される．

```
procedure subtranspose(source, destination, k, l, m, mask);
  array source, destination, mask;
  integer k, l, m, q, u, v, w;
  for u := 0 step 1 until m − 1 do
    for w := 0 step 1 until k − 1 do
```

```
for v := 0 step 1 until m − 1 do
  begin
    clearregister(1);
    for q := 1 step 1 until l − 1 do
      begin
        register[0] := source[k × l × u + k × q + w];
        register[0] := register[0] ∧ mask[q];
        shiftright(0, k × (l − 1 − q);
        register[1]; = register[1] ∨ register[0]
      end;
    destination[k × l × u + k × (l − 1 − v) + w] := register[1]
  end
```

ここで，clearregister(1) は register[1] の内容を０にする命令である．また，mask[q] は subrotate の場合と同様に $2^{k(\ -1-q)}$（2^k-1）（2^n-1）／（2^k-1）である．

ｎ×ｎのドットパターンの１８０°の回転は，$n=k_1k_2\cdots k_t$ と分解した上で，次のステップ１，ステップ２，…，ステップｔを実行すればよい．

ステップ１　subtranspose (source, work2, 1, k_1, $k_2\cdots k_t$, mask1)

ステップ２　subtranspose (work2, work3, k_1, k_2, $k_3\cdots k_t$, mask2)

⋮

ステップｔ　subtranspose (workt, destination, $k_1\cdots k_{t-1}$, k_t, 1, maskt)

ただし，work2, work3, …, workt, mask1, mask2, …, maskt はR_{90}におけるものと同じである．

R_{180}の手間は Ｏ（（$k_1+k_2+\cdots+k_t$）ｎ）である．とくに，$n=2^t$ のときは，含まれる命令の数は

ＭＯＶＥ命令	４ｎ $\log_2$ ｎ 回
ＡＮＤ命令	２ｎ $\log_2$ ｎ 回
ＯＲ命令	２ｎ $\log_2$ ｎ 回
ＳＨＩＦＴ命令	２ｎ $\log_2$ ｎ 回

であり，手間はＯ（ｎ log ｎ）である．ｎ＝３２のときにＭＣ６８０００で実行すると３．６５６ミリ秒となる．

２７０°の回転はR_{90}を３回行ったり，R_{180}とR_{90}を１回ずつ行うのでなく，－９０°の回転を行うのがよい．それには，R_{90}における

$$shiftright(q, k \times (q - v))$$

を

$$shiftright(q, k \times (v - q))$$

で置き換えればよい．これをR_{270}とする．R_{270}の手間はR_{90}と同じである．

５．ドットパターンの回転・反転等のなす群

次の式は容易に確かめられる．

$R_{90}R_{90}=R_{180}$

$R_{90}R_{180}=R_{180}R_{90}=R_{90}R_{90}R_{90}=R_{270}$

$R_{90}R_{270}=R_{270}R_{90}=R_{180}R_{180}=I$

ただし，Ｉは恒等変換である．

この外，ドットパターンを左右，上下に対称に反転する操作も有用である．それらをＨ，Ｖと記すことにしよう．こうして，次の群Ｇが得られる（図　）．

$G=\{I, R_{90}, R_{180}, R_{270}, H, VR_{270}, V, VR_{90}\}$

左右反転Ｈは，subtranspose における

$$destination[k \times l \times u + k \times (l - 1 - v) + w] := register[1]$$

を

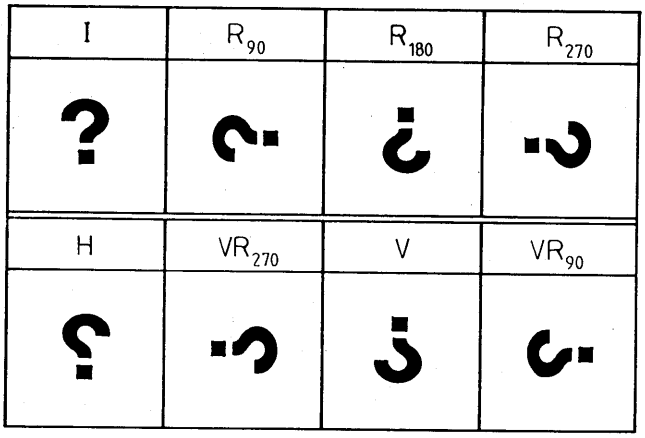

図5　ドットパターンの回転・反転等のなす群

$$destination[k \times l \times u + k \times v + w] := register[1]$$

で置き換えたものを用いてR180を行えばよい．したがって，Hの手間はR180と同じである．

上下反転Vの手順は自明であろう．手間はO（n）で，R90，R180，R270，H等と比べれば無視できる．したがって，VR270，VR90 のために新しいアルゴリズムを考える必要はなく，Vと R270やR90 を組み合わせればよい．

以上から，R90，R180，R270，H，V だけのプログラムを作ればよい．

6．高速フーリエ変換との関係

アルゴリズムR90と高速フーリエ変換（ＦＦＴ）とを比較するのは興味深い．一般に，フーリエ変換は

$$\sigma(f) = \int_0^{2\pi} e^{-ift} x(t)\, dt,$$

で定義される．ただし，ｘ（ｔ）は信号，σ（ｆ）はスペクトラムである．

数値計算では，次式によって計算する．

$$\hat{\sigma}(f) = \sum_{t=0}^{n-1} \omega^{ft} x\Big(\frac{2\pi}{n} t\Big) \qquad (f = 0, 1, \ldots, n-1)$$

ただし，

$$\omega = exp\,\Big(-\frac{2\pi i}{n}\Big) = cos\,\frac{2\pi}{n} - i\,sin\,\frac{2\pi}{n}$$

上式にしたがってσ（ｆ）を計算すると，手間はO（n²）かかる．高速フーリエ変換は次の原理にしたがってσ（ｆ）を計算する．

ｎ＝ｋℓｍとし，ｆ＝ℓｍζ＋ｍη＋θ，ｔ＝ｋℓｐ＋ｋｑ＋ｒとする．

$$\begin{aligned}\hat{\sigma}(f) &= \sum_{r=0}^{k-1} \sum_{q=0}^{l-1} \sum_{p=0}^{m-1} \omega^{(lm\varsigma + m\eta + \theta)(klp + kq + r)} x\Big(\frac{2\pi}{n}(klp + kq + r)\Big) \\ &= \sum_{r=0}^{k-1} \omega^{(lm\varsigma + m\eta + \theta)r} \sum_{q=0}^{l-1} \omega^{(m\eta + \theta)kq} \sum_{p=0}^{m-1} \omega^{\theta klp} x\Big(\frac{2\pi}{n}(klp + kq + r)\Big)\end{aligned}$$

そこで，次の部分フーリエ変換ｓｕｂｆｔを考える．

```
procedure subft(signal, spectrum, k, l, m, omega);
  array signal, spectrum;
  integer k, l, m, p, q, r;
  complex omega;
  begin
```

```
        w := 0;
        for theta := 0 step 1 until m − 1 do
          for eta := 0 step 1 until l − 1 do
            for r := 0 step 1 until k − 1 do
              for q := 0 step 1 until l − 1 do
                w := w + omega ↑ ((m × eta + theta) × k × q)×
                          signal[k × l × θ + k × q + r];
              spectrum[k × l × theta + k × q + r] := w
    end
```

高速フーリエ変換は，n＝$k_1 k_2 \cdots k_t$と分解した上で，次のステップ１，ステップ２，…，ステップｔをこの順に実行すればよい．

ステップ１　subft (signal, work2, $k_1 \cdots k_{t-1}$, k_t, 1, omega)

ステップ２　subft (work2, work3, $k_1 \cdots k_{t-2}$, k_{t-1}, k_t, omega)

⋮

ステップｔ　subft (workt, spectrum, 1, k_1, $k_2 \cdots k_t$, omega)

手間は Ｏ（（$k_1 + k_2 + \cdots + k_t$）n）である．とくに，$n = 2^t$ のときは，Ｏ（n log n）である．

７．まとめ

ｎ×ｎドットパターンを９０°，１８０°，２７０° 回転したり，左右・上下に反転したりする演算がなす群の構造を考察し，それらをＯ（n log n）の手間で行うアルゴリズムを提案した．とくに，ｎ＝３２の場合に，ＭＣ６８０００で５ミリ秒以内でできることも示した．また，それらのアルゴリズムと高速フーリエ変換との類似点についても考察した．

ｎが語長より大きい場合にもこのアルゴリズムを利用することはできるが，その場合は，ｎ×ｎドットパターンを記録するデータ構造を工夫する方が重要であろう．その問題については，別の機会に論ずることにする．

謝辞

本研究のきっかけを与えて下さった東京農工大学工学部数理情報工学科助手中川正樹氏，同大学院学生関口治氏（現㈱日立製作所）に感謝申し上げます．

参考文献

[1] H. Samet, “The quadtree and related hierarchical data structure,” ACM Computing Surveys 16, 187-260 (1984).

[2] MC68000 User’s Manual, Motorola Co. Ltd., 1979.

[3] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” Mathematics of Computation 19, 297-301 (1965).