ビット・ベクトルとインダイレクト・ベクトルを持つパイプライン計算機上の FFT のための新しい並列算法†

平岩 健三††

本論文では，パイプライン計算機（ベクトル計算機）によって FFT（高速フーリエ変換）を行うための新しい並列算法が議論される．

この算法は modified recursive doubling 技法に基づいた W 表の作成，Tuttle の算法から導びかれたビット・ベクトルとインダイレクト・ベクトルを用いた変換，及びインダイレクト・ベクトルを用いたビット反転の3部によって構成される．そしてこの算法の演算量が評価される．

さらにこの算法が実際のパイプライン計算機によって実行され，そしてその処理時間と結果の精度が，通常の計算機によるそれらと比較される．その結果としてこのビット反転の部分は通常の計算機によって処理されたとしても，Singleton のそれより速いことも明らかにされる．

1. はじめに

最近，膨大な計算時間を要する問題が計算機の処理対象となってきた．そのような問題の対処の一手段として，1960年代から並列計算機が開発されている[3),8),12)]．これはデータ並びの規則性に着目し，そのデータ群をベクトル演算的に実行するもので，アレイ計算機（ILLIAC IV 型，以降 A.C. と略す）とパイプライン計算機（CDC-STAR 型，ベクトル計算機とも言われ，以降 P.C. と略す）に大別されている．本論文では P.C. を基盤におき，アプリケーション分野で重要なフーリエ変換の並列計算機上の処理を考察する．

FFT (Fast Fourier Transform) は，フーリエ変換の高速処理技法として，1965年に Cooley-Tukey により提案され[14)]，そしてそのハードウェア化が指摘されると共に，小型専用プロセッサとしてハードウェア化の研究と実用化がなされてきた[2)]．一方並列計算機上のソフトウェアとしては ILLIAC IV 計算機による FFT の実行報告がある[3)]．しかしこの並列算法は P.C. に向いていない．また Tuttle は P.C. による正順入力逆順出力 FFT を行うための並列算法を提案しているが，これは変換部分のみを扱う並列化処理技法であり[4)]，これだけでは十分な効率向上を望めない．

本論文では FFT のための新しい並列算法を説明するため，最初に P.C. を想定し，その基本的性質，機能などについて述べ，後の議論で必要となる通常の FFT に関する必要最小限の概念を述べる．次にこの P.C. によって FFT を行うための並列算法を提案する．この算法は W 表の作成，ビット・ベクトルとインダイレクト・ベクトルを用いた変換，及びインダイレクト・ベクトルを用いたビット反転の3部から成っている．そしてこの算法の演算量を評価する．

さらにこの算法を実際の P.C. によって実行し，その処理時間と結果の精度を，通常の計算機によるそれらと比較する．その結果として，このビット反転の部分は，通常の計算機によって処理されたとしても，Singleton のそれより速いことも明らかにする．

2. 準備

本章では並列 FFT 算法を明確にするために2つの準備をする，まず想定されるパイプライン計算機について議論し，次に通常の FFT について簡単に述べる．また，これらの中で記号の説明を行う．

2.1 パイプライン計算機 (P.C.)

2.1.1 データ形式とデータ種類

この P.C. が扱うデータ形式は，ビット，整数型，及び実数型の3形式である．ビットデータの単位は1ビット，整数型・実数型データの単位は1語である．

また，この P.C. が扱うデータ種類は，スカラ，ベクトル（以降 VCT と略す），インダイレクト・ベクトル）以降 IVCT と略す），及びビット・ベクトル（以降 BVCT と略す）の4種である．スカラとは単一データである．VCT とは語単位で等間隔 d（整数）に

† A New Parallel Algorithm for the Fast Fourier Transform on Pipeline Computers with both Bit Vector and Indirect Vector Facilities by KENZO HIRAIWA (FSC Department, Software Division, Fujitsu Ltd.).

†† 富士通(株)ソフトウェア事業部フィールドサポートセンタ (FSC) 部

並んだデータ群である．IVCT とは VCT の内容で示されるアドレスに格納されたデータ群である．そして BVCT とはビット単位で等間隔に並んだデータ群である．VCT, IVCT, BVCT を総称して，一般的にベクトルとも呼ぶ．

2.1.2 P.C. のベクトル演算

e, e', e''（整数）を始点として，等間隔 d, d', d'' に並んでいるいずれもN個の要素から成る3個のベクトルを $(a_{dv+e}), (b_{d'v+e'}), (c_{d''v+e''})$, $v=0, 1, \cdots, N-1$ とする．そのとき通常の計算機が

$$a_{dv+e} \leftarrow b_{d'v+e'} \text{ op } c_{d''v+e''}, \quad v=0, 1, \cdots, N-1 \tag{1}$$

をNオペレーションで処理するのに対し，並列計算機のベクトル演算はこれらをオペランドとして概念的には1オペレーションでこれを処理する．なお A.C. では並列的に，P.C. では時分割的にこれを行う．しかし一般に並列計算機ではベクトル演算を行う前にすべてのオペランドは定義されていなければならない*．また，(1)をベクトル演算するとき，式及び添字を

$$a(dv+e)=b(d'v+e') \text{ op } c(d''v+e'')$$
$$0 \leq v \leq N-1 \tag{2}$$

のように記述し，それを明示することにする．

ベクトル演算は，VCT, IVCT で示される2種類の整数型・実数型データ群に関して四則演算出来る．また，BVCT で示されるビットデータ群に関する移動処理が出来る．そして補助的なスカラ演算も出来る．

2.1.3 ベクトル演算の時間式の特徴とその表現

P.C. のベクトル演算の時間 T は，粗く言って以下の一次式で表わされる．

$$T=PN+R \quad P>0, \ R>0, \ N \geq 1 \tag{3}$$

ここにNは要素数である．また，Pを各ベクトル演算に要する基本単位時間，Rを立ち上り時間と呼ぶ．また，同じ機能を有するスカラ演算の時間をSとすると

$$P<S<P+R \tag{4}$$

が成立する．(3), (4)から十分大きな要素数 N に対して時間に関する不等式

$$PN+R<SN \tag{5}$$

が成立する．そして通常 $P<R$ が成立する．

なお，P と R は四則演算によって一般に異なるので，$+, -, *, /$ をPとRの右下に付けて示す[5),7)]（$P_+, P_-, P_*, P_/, R_+, R_-, R_*, R_-$）．整数型・実数型データ群の移動処理は0を加えると考える．整数型・実数型データ群とビットデータ群の移動処理を区別するためにビットデータ群の場合 B を右上に付けて示す．また，IVCT を少なくとも1つのオペランドに用いた場合 I を右上に付けて示す．例えば $P_+{}^I N+R_+{}^I$ は，少なくとも1つのオペランドに IVCT を用いた要素数 Nのベクトル加算の時間を示す．

2.2 通常の計算機による FFT 算法

本節では，通常の計算機による2基底の FFT について述べる．離散型フーリエ変換は $X_0(k)$, $k=0, 1, \cdots, N-1$ なるN個の複素データから，$X(n)$, $n=0, 1, \cdots, N-1$ なるN個の複素データを求めるものであり，それは

$$X(n)=\sum_{k=0}^{N-1} X_0(k)\omega^{kn} \tag{6}$$

$n=0, 1, \cdots, N-1$

$\omega=\exp(2\pi i/N)$ (i: 虚数単位)

と定義される（正規化定数は省略する）．そのときフーリエ逆変換は

$$X_0(k)=\sum_{n=0}^{N-1} X(n)\omega^{-kn} \tag{7}$$

で定義される．今，Nが2の巾で表わされるとし，$m=\log_2 N$ とすると

$$k=k_{m-1}2^{m-1}+k_{m-2}2^{m-2}+\cdots+k_2 2^2+k_1 2^1+k_0 \tag{8}$$

$$n=n_{m-1}2^{m-1}+n_{m-2}2^{m-2}+\cdots+n_2 2^2+n_1 2^1+n_0 \tag{9}$$

となり，(8), (9)の記号を用いると，(6)は

$$\begin{aligned}
&X_1(n_0, k_{m-2}, \cdots, k_2, k_1, k_0)\\
&\quad=\sum_{k_{m-1}=0}^{1} X_0(k_{m-1}, k_{m-2},\\
&\qquad \cdots, k_2, k_1, k_0)\omega^{n_0(k_{m-1}2^{m-1}+\cdots+k_1 2+k_0)}\\
&X_2(n_0, n_1, \cdots, k_2, k_1, k_0)\\
&\quad=\sum_{k_{m-2}=0}^{1} X_1(n_0, k_{m-2},\\
&\qquad \cdots, k_2, k_1, k_0)\omega^{2n_1(k_{m-2}2^{m-2}+\cdots+k_1 2+k_0)}\\
&\cdots\cdots\\
&X_m(n_0, n_1, \cdots, n_{m-3}, n_{m-2}, n_{m-1})\\
&\quad=\sum_{k_0=0}^{1} X_{m-1}(n_0, n_1, \cdots, n_{m-3}, n_{m-2}, k_0)\omega^{n_{m-1}2^{m-1}k_0}
\end{aligned} \tag{10}$$

$$X(n_{m-1}, n_{m-2}, \cdots, n_2, n_1, n_0)=X_m(n_0, n_1, \cdots, n_{m-3}, n_{m-2}, n_{m-1}) \tag{11}$$

と書くことが出来る．(10)はSande-Tukey algorithm と呼ばれ，(11)はビット反転と呼ばれる．ここに X_1,

* 例えば通常の計算機による漸化式の計算などは，オペランドの再定義が起こるため1ベクトル演算では行えず，普通並列計算機向きでない．

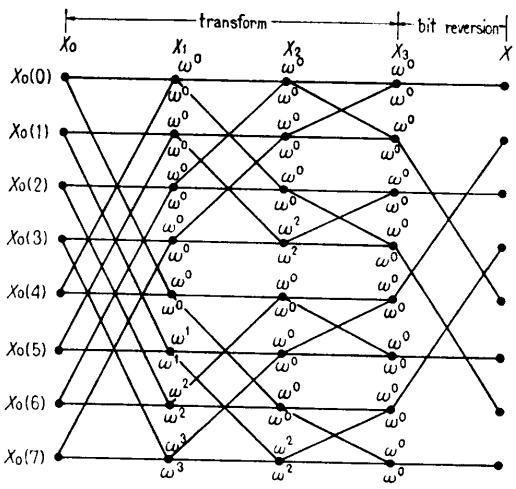

図 1 Sande-Tukey algorithm によるデータの流れ
Fig. 1 Data flow by Sande-Tukey algorithm.

$X_2, \cdots, X_m$ は変換の各 stage を示している．$N=8$ の場合の (10), (11) によるデータの流れと ω^r ($r=0, 1, 2, 3$) を**図 1** に示す．

3. パイプライン計算機による新しい FFT 算法

3.1 W 表作成のための並列算法

2.2 節で述べた ω の巾乗の集りを通常の計算機によって高速に求めるためには種々の算法がある[1]．これらはいずれも漸化式から成るものが多く，例えば座標回転による算法は

$$\begin{cases} \cos(l+1)\theta = \cos\theta\cos l\theta - \sin\theta\sin l\theta \\ \sin(l+1)\theta = \sin\theta\cos l\theta + \cos\theta\sin l\theta \end{cases} \quad (12)$$

$$l=1, 2, \cdots, N/8-2, \quad \theta=2\pi/N$$

に依拠している．これを順次 Cox 法と呼ぶことにする．

しかし漸化式は 2.1.2 項の脚注＊から P.C. による効率向上を期待出来ない．それ故，従来の P.C. による FFT に関する論文は，これの集りをあらかじめ別の方法で計算されたものとして扱うのが普通である．そこでここでは P.C. によってこれの集りを効率良く求める算法を述べる．

今，余弦と正弦の入る 2 つの大きさ $N/2$ の実数型一次元配列をまとめて W とし，$W(r)=\omega^r$, $r=0, 1, \cdots, N/2-1$ が入るものとする．そしてこれを W 表と呼ぶことにする．

W 表の作成は ω の巾乗の集合 $\{\omega^1, \omega^2, \omega^3, \cdots, \omega^{N/8-1}\}$ を求めることを主要な目的とするが，これを並列処理によって効率的に求めるためには，次のように行えば良い．今もし，ω^1 とベクトル $(\omega^2, \omega^4, \omega^6, \cdots, \omega^{N/8-2})$ が得られていると仮定すれば，このベクトルの ω^1 によるスカラ倍はベクトル $(\omega^3, \omega^5, \omega^7, \cdots, \omega^{N/8-1})$ を生ずる．すなわち差集合 $\{\omega^3, \omega^5, \omega^7, \cdots, \omega^{N/8-1}\} = \{\omega^1, \omega^2, \omega^3, \cdots, \omega^{N/8-1}\} - \{\omega^2, \omega^4, \omega^6, \cdots, \omega^{N/8-2}\} - \{\omega^1\}$ の元を成分にもつベクトルを得ることが出来る．同様に ω^2 と $(\omega^4, \omega^8, \omega^{12}, \cdots, \omega^{N/8-4})$ との積によって $(\omega^6, \omega^{10}, \omega^{14}, \cdots, \omega^{N/8-2})$ が導びかれる．上述の議論を繰り返すことにより，目的の集合の元を成分にもつベクトルを得ることが出来る．これは VCT 演算を約 $\log_2 N$ 回行うことによって完了する．上述の処理は modified recursive doubling 技法[6]に基づき，それを改良したものである．この処理を並列 Cox 法と呼ぶことにする．並列 Cox 法は VCT 演算を使用するため，$j=1, 2, \cdots, m-4$ に関して大きさ $2^{m-3-j}-1$ の作業領域 ($W^{(j+1)}$ と記す) を必要とする．便宜上 $W(l)=W^{(1)}(l-1)=\omega^l$, $l=1, 2, \cdots, N/8-1$ とし，$\{\omega^1, \omega^2, \omega^3, \cdots, \omega^{N/8-1}\}$ を求めるためのこの算法の実際の手順は次のようである．

始めに $W^{(1)}(0)=\omega^1$ を初期値として与える．次に

$$W^{(j+1)}(0) = W^{(j)}(0)W^{(j)}(0),$$
$$j=1, 2, \cdots, m-4, \quad (m=\log_2 N) \quad (13)$$

によって $\{\omega^2, \omega^4, \omega^8, \cdots, \omega^{N/16}\}$ を計算する．さらに

$$\begin{cases} W^{(m-3-j')}(2p+1) = W^{(m-2-j')}(p) & (14) \\ W^{(m-3-j')}(2p+2) \\ \quad = W^{(m-3-j')}(2p+1)W^{(m-3-j')}(0) & (15) \end{cases}$$
$$0 \le p \le 2^{j'}-2, \quad j'=1, 2, \cdots, m-4$$

を計算することによって，直ちに $\{\omega^1, \omega^2, \omega^3, \cdots, \omega^{N/8-1}\}$ を得る．このとき $W^{(1)}(0), W^{(1)}(1), W^{(1)}(2), \cdots, W^{(1)}(N/8-2)$ の各々には $\omega^1, \omega^2, \omega^3, \cdots, \omega^{N/8-1}$ が格納されていることになる．ここで計算途中で表われる $W^{(1)}(0)$, $W^{(2)}(0), W^{(3)}(0), \cdots, W^{(m-3)}(0)$ には $\omega^1, \omega^2, \omega^4, \cdots, \omega^{N/16}$ が格納されているので，これらの量の重複計算の必要はない．したがって，その分だけ演算量の節約が計れる．

ここに (13) はスカラ演算によって行われるが，並列 Cox 法の主要部は VCT 演算によって行われる (14), (15) の計算であることに注意する．なお $W(0)=\omega^0$, $W(N/8)=\omega^{N/8}$ は容易に求まるため，その議論は省略した．$N/8-1=7$ の場合の (13), (14), (15) によるデータの流れを**図 2** に示す．この算法を $\{\omega^0, \omega^1, \omega^2, \cdots, \omega^{N/8}\}$ に適用し，残り $\{\omega^{N/8+1}, \omega^{N/8+2}, \omega^{N/8+3}, \cdots, \omega^{N/4}\}$, $\{\omega^{N/4+1}, \omega^{N/4+2}, \omega^{N/4+3}, \cdots, \omega^{N/2-1}\}$ は，作業領域上に加法公式を用いて求める．

3.2 Tuttle の並列変換

2.2 節で述べた FFT の変換部分の並列処理の技法はすでに Tuttle ら*によって提案されており，その特徴は通常必要とされる各 stage でのアドレス計算を(10)の各 stage で参照格納する X_{i-1}, X_i の要素アドレスの変更によって代行させている点である．しかしここでは Tuttle らの算法がまだ一般によく知られていないことと，次節で提案される新しい手法との比較において必要とされることから，その骨子を以下に述べておく．各 stage を書き出すと以下のようである．

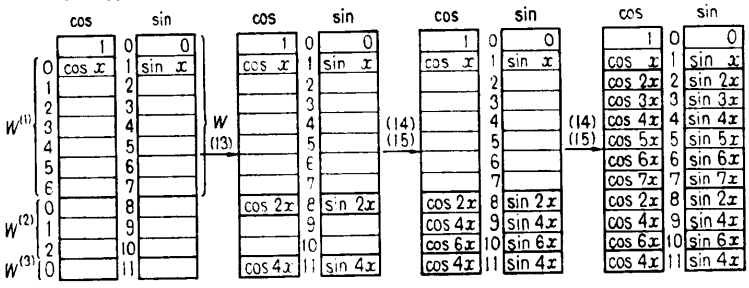

図 2　W 表作成の並列算法
Fig. 2　Parallel algorithm of W (cos/sin) table generation.

$$
\begin{aligned}
&X_1(k_{m-2}, k_{m-3}, \cdots, k_1, k_0, n_0)\\
&\quad=\sum_{k_{m-1}=0}^{1} X_0(k_{m-1}, k_{m-2},\\
&\qquad \cdots, k_2, k_1, k_0)\omega^{n_0(k_{m-1}2^{m-1}+\cdots+k_1 2+k_0)}\\
&X_2(k_{m-3}, k_{m-4}, \cdots, k_0, n_0, n_1)\\
&\quad=\sum_{k_{m-2}=0}^{1} X_1(k_{m-2}, k_{m-3},\\
&\qquad \cdots, k_1, k_0, n_0)\omega^{2n_1(k_{m-2}2^{m-2}+\cdots+k_1 2+k_0)}\\
&\cdots\cdots\\
&X_m(n_0, n_1, \cdots, n_{m-3}, n_{m-2}, n_{m-1})\\
&\quad=\sum_{k_0=0}^{1} X_{m-1}(k_0, n_0,\\
&\qquad \cdots, n_{m-4}, n_{m-3}, n_{m-2})\omega^{n_{m-1}2^{m-1}k_0}
\end{aligned}
\tag{16}
$$

第 i stage で必要とされる W 表を W_{i-1} で表わすとすれば，W_{i-1} は(16)から

$$
W_{i-1}(r)=\omega^{[r/2^{i-1}]2^{i-1}},\quad r=0,1,\cdots,N/2-1,\ i=1,\cdots,m \tag{17}
$$

と書けるので，第 i stage での(16)，(17)は簡単な計算によって，次のような算法となる．

$$
\begin{cases}
X_i(2r)=X_{i-1}(r)+X_{i-1}(r+N/2) & 0\le r\le N/2-1\\
X_i(2r+1)=W_{i-1}(r)\{X_{i-1}(r)-X_{i-1}(r+N/2)\} & 0\le r\le N/2-1\\
W_i(2r)=W_{i-1}(r)W_{i-1}(r) & 0\le r\le N/4-1\\
W_i(2r+1)=W_i(2r) & 0\le r\le N/4-1
\end{cases}
\tag{18}
$$

ここに $i=1,2,\cdots,m$．(17)の[ ]はガウス記号．したがって(18)は r に関して VCT 演算可能であることを示している．また，(18)から X_{i-1} と X_i の要素アドレスを参照格納するための計算を必要としないことが判る．$N=8$ の場合の(18)によるデータの流れを**図3**に，W 表の変化を**図4**に示す．この算法を便宜的に Tuttle の算法と呼ぶことにする．この算法が並列処理において Sande-Tukey algorithm よりも優れていることを付録に示す．

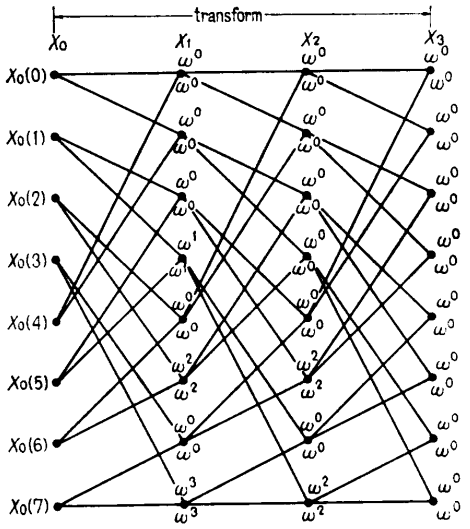

図 3　Tuttle の算法によるデータの流れ
Fig. 3　Data flow by Tuttle algorithm.

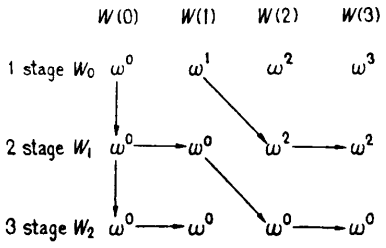

図 4　各 stage における W 表の変化
Fig. 4　Transition of W table at each stage.

3.3 BVCT と IVCT を用いた並列変換

Tuttle の算法は，十分な並列性を有する優れたものではあるが，ここではさらに改善された並列算法を提案する．3.2 節では VCT 加減乗算を用いる算法に

* この並列算法は，独立に何人かの研究者によって発展させられたようである[15]．

よって，W 表が各 stage において変化していることを示している．しかしここで提案する算法は，さらにその加減乗算を必要としない算法である．その説明のためにまず BVCT と IVCT を用いて，W 表の内容の取り出し方を考える．

HをW表への指示ベクトルとし，大きさ $N/2$ の整数型一次元配列とする．$H(r)$，$r=0,1,\cdots,N/2-1$ には各々 r が割り当てられるとすると，これは丁度 W 表作成直後の ω の巾指数に対応している．Hの任意の要素をrとし，これを2進 $m-1$ 桁の数とみなすと

$$r=k_{m-2}2^{m-2}+\cdots+k_2 2^2+k_1 2^1+k_0$$

と表わすことが出来る．そのとき第 i stage の変換において，(18)で必要とする ω の巾指数を r_i で示すと，(17)から

$$r_1=k_{m-2}2^{m-2}+\cdots+k_2 2^2+k_1 2+k_0$$

$$\cdots\cdots$$

$$r_{i-1}=k_{m-2}2^{m-2}+\cdots+k_{i-1}2^{i-1}+k_{i-2}2^{i-2}$$

$$r_i=k_{m-2}2^{m-2}+\cdots+k_{i-1}2^{i-1}$$

$$\cdots\cdots$$

$$r_m=0$$

となっている．これらは次のような漸化式で表わすことが出来る．

$$r_1=r$$

$$r_i=r_{i-1}-k_{i-2}2^{i-2}\quad i=2,\cdots,m. \tag{20}$$

(20)を用いて r_i を求めるためには，初期値として，Hに0から $N/2-1$ までの数列を入れておき*，(18)の前半2式を行ったあと，第 1 stage では1桁目の k_0 の $N/2$ 個のビット列を $N/2$ 個の0ビット列で置き換え，第 2 stage では2桁目の k_1 を0ビット列で置き換え（k_0 はすでに0なっている），以下同様に第 $m-1$ stage では k_{m-2} を0ビット列で置き換える．これは BVCT の移動処理によって容易に行うことが出来る．こうしてW表への指示ベクトルHをビット操作で修正しつつ，Hをインデックスとして，IVCT を使用し W 表の内容を取り出して，(18)の前半2式を行えば良い．また，この算法は明らかに W 表そのものを変えていないことが判る．こうして W 表に丸めの誤差を累積させないことが可能である．

$N=8$ の場合のHの変化とデータの流れを**図5**に示す．この例では，1語3ビットとすると，第 1 stage では0語目の2ビット目 (k_0) から，第 2 stage では1ビット目 (k_1) から，第 3 stage では 0 ビット目 (k_2) から，各々3ビットおきに4ビット0を移動している．

3.4 IVCT を用いた並列ビット反転

P. C. によるビット反転のための処理も重要である．そこでここではそのための並列算法を述べる．これは主に VCT と IVCT を用いて実現する．(11)から最後の変換を受けた $X_m(n^v)$ に番号を付け，それをn^vとし2進m桁で表わすと

$$n^v=n_0 2^{m-1}+n_1 2^{m-2}+\cdots+n_{m-3}2^2+n_{m-2}2+n_{m-1} \tag{21}$$

となる．並べ換え（ビット反転）とは

$$n=n_{m-1}2^{m-1}+n_{m-2}2^{m-2}+\cdots+n_2 2^2+n_1 2+n_0 \tag{22}$$

なる $X(n)$ に $X_m(n^v)$ を移し換えることである．

この n^v を計算するために，(21)で用いた記号を使って以下に n^i を定義する．

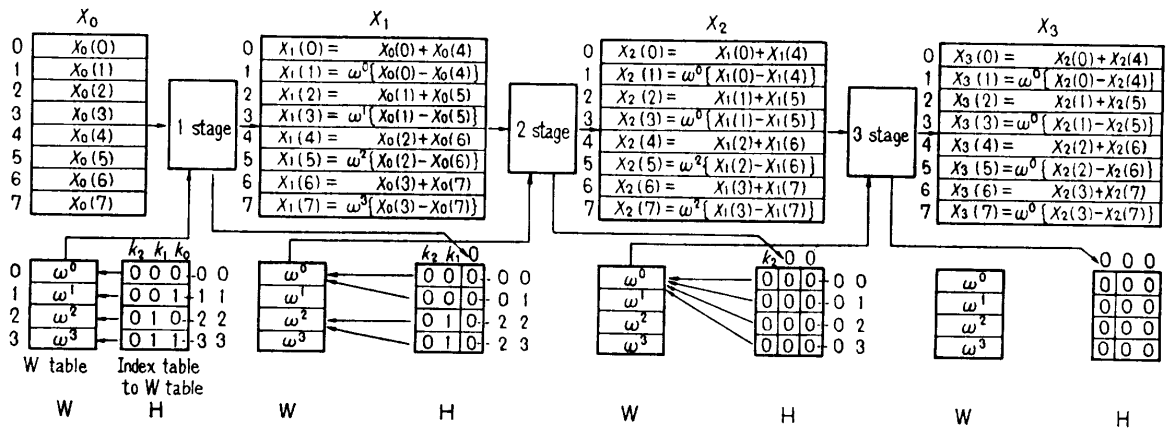

図 5 ビット・ベクトルとインダイレクト・ベクトルを用いた並列変換
Fig. 5 Parallel algorithm of transform by means of bit and indirect vectors.

* 数列発生の並列算法は，3.4 節で述べるものと本質的に同じである．演算量は少ないので無視出来る．

$$n^i=\sum_{j=m-i}^{m-1} n_{m-j-1}2^j$$

n_{m-j-1} は 0 または 1, $i=1,\cdots,m$.　(23)

これを展開して示せば

$$n^1=n_0 2^{m-1}$$
$$\cdots\cdots$$
$$n^{i-1}=n_0 2^{m-1}+n_1 2^{m-2}+\cdots+n_{i-2}2^{m-i+1}$$
$$n^i=n_0 2^{m-1}+n_1 2^{m-2}+\cdots+n_{i-2}2^{m-i+1}+n_{i-1}2^{m-i}$$
(24)
$$\cdots\cdots$$
$$n^m=n^{\nu}=n_0 2^{m-1}+n_1 2^{m-2}+\cdots$$
$$+n_{m-3}2^2+n_{m-2}2+n_{m-1}$$

である．(24)は次のように整理することが出来る．

$$n^i=\begin{cases} n^{i-1} & \text{if } n_{i-1}=0 \\ n^{i-1}+2^{m-i} & \text{if } n_{i-1}=1 \end{cases} \quad (25)$$
$$i=2,\cdots,m$$

したがって上式は n^{i-1} が既知ならば n^i は容易に求まることを示している．

n^i は次のように並列計算出来る．ADX をその作成過程で n^i が入る X_m への指示ベクトルとし，大きさNの整数型一次元配列とする．ここに $ADX(n)$, $n=0,1,\cdots,N-1$ には n^{ν} が最終的に割り当てられるとすると，(25)は以下の漸化式で書ける．

$$ADX(0)=0$$
$$ADX(p+2^{i-1})=ADX(p)+2^{m-i} \quad (26)$$
$$0\leq p\leq 2^{i-1}-1,\quad i=1,\cdots,m.$$

ここに $X_m(0)$ は $X(0)$ に移り変わるため，$ADX(0)$ は初期値 0 とすればよい．(26)は ADX が p に関する VCT 加算によって求めることが出来ることを示している．こうして作成された X_m への指示ベクトル ADX をインデックスとして，IVCT を用いて X_m からXへデータを 1 オペレーションで並べ換えることが出来る．この算法の $N=8$ の場合のインデックスの作成の流れと並べ換えの流れを**図 6** に示す．

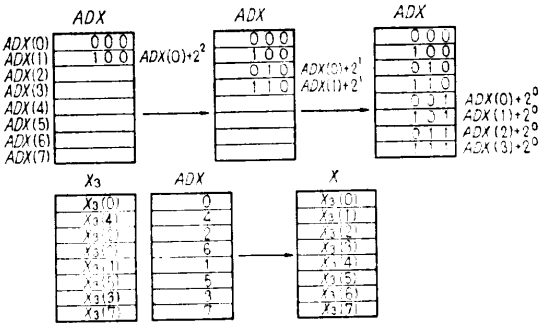

図 6 インダイレクト・ベクトルを用いた並列ビット反転
Fig. 6 Parallel algorithm of bit reversion by means of indirect vector.

4. 各算法の演算量について

通常の計算機による FFT は種々のものがあり，一般的に実際の演算量を比較すると複雑になるので，ここでは前述の通常の計算機による FFT 算法とそれを基盤にして並列化した P.C. による FFT 算法の演算量を比較する．すなわち W 表作成では順次 Cox 法と並列 Cox 法，変換では Sande-Tukey algorithm と Tuttle の算法と 3.3 節の算法，ビット反転では並列算法を順次算法に変えたものの演算量を比較する．なお 2.2 節と 3 章で複素型として扱ったものについては，ここでは実数型に直して議論している．

さて $\{\omega^0,\omega^1,\omega^2,\cdots,\omega^{N/8}\}$ の W 表作成の順次 Cox 法の演算量は，(12)から

$$(4S_*+1S_++1S_-)N/8+O(1) \quad (27)$$

となり，並列 Cox 法の演算量は，(13)から

$$(4S_*+1S_-)\log_2 N+O(1) \quad (28)$$

また，後半では(14)と(15)から，要素数の変化は等比数列になることに注意すると

$$(4P_*+1P_++1P_-+2P_=)N/8+$$
$$(4R_*+1R_++1R_-+2R_=)\log_2 N-$$
$$(4P_*+1P_++1P_-)\log_2 N+O(1) \quad (29)$$

となる．これにW表の残りの部分，すなわち$\{\omega^{N/8+1},\omega^{N/8+2},\omega^{N/8+3},\cdots,\omega^{N/4}\}$，$\{\omega^{N/4+1},\omega^{N/4+2},\omega^{N/4+3},\cdots,\omega^{N/2-1}\}$ を求めるための演算量を加え合わせたものを**表 1** に示す．同様に変換，ビット反転の演算量も表 1 に示す．

表 1 を基に種々の考察を行う．まず W 表作成のための順次 Cox 法と並列 Cox 法を比較する．後者は前者よりも移動処理の演算量が多いので，いちがいに後者は前者よりも速いと結論出来ない．しかし $\log_2 N$ 以下の項を無視して，$6S_=>8P_=$ を仮定すると（この仮定は現実の P.C. では十分妥当性を持つものではあるが），十分大きなNに対して並列 Cox 法は順次 Cox 法よりも速くなることを示している．また，変換，ビット反転共，十分大きなNに対して並列算法は順次算法より問題なく速くなることが判る．

一方並列変換同志の比較，すなわち Tuttle の算法と 3.3 節の算法では，$\log_2 N$ の項を無視して共通項を差し引くと

$$6P_*+0.5P_-+1P_= \quad (30)$$

と

表 1 順次 FFT 算法と並列 FFT 算法との演算量の比較
Table 1 Comparison of the number of operations of serial vs parallel algorithms of FFT.

serial/parallel / module	serial algorithm	parallel algorithm
W table generation	$(4S_*+1S_+ +1S_- +6S_=)(N/8)+O(1)$	$(4P_*+1P_+ +1P_- +8P_=)(N/8)+(4R_*+1R_+ +1R_- +2R_= +4S_* +1S_- -4P_* -1P_+ -1P_-)\log_2 N+O(1)$
transform	$(6S_*+3S_+ +3.5S_- +1S_=)(N\log_2 N/2)$	Tuttle algorithm $(6P_*+3P_+ +3.5P_- +1P_=)(N\log_2 N/2)+(8R_*+3R_+ +4R_- +2R_=)\log_2 N$ New parallel algorithm $(4P_*^I+3P_+ +3P_- +1P_=^B)(N\log_2 N/2)+(4R_*^I+3R_+ +3R_- +1R_=^B)\log_2 N$
bit reversion	$(1S_+ +6S_= +T)N-6S_=2^{[(1+\log_2 N)/2]}$ T: test and/or branch operation times [ ]: Gaussian symbol	$(1P_+ +2P_= +2P_=^I)N+R_+\log_2 N+O(1)$

Where S: scalar operation time, P: basic unit time for each vector operation, R: start up time for each vector operation.
$O(1)$: order of unity term, N: complex data points.

$$4P_*^I+1P_=^B \tag{31}$$

の小さい方が速いことを示している. (30), (31) の量は種々の P. C. の処理速度により決定されるが, P. C. の特性を考えると実現可能な仮定と思われる $P_*=P_*^I$, $P_=P_=^B$ とする. こうすると $2P_*+0.5P_-$ だけ, 3.3 節の算法が速くなる.

5. 実験と考察

現在までのところ 2.1.1 項で述べたデータ種類を十分に満足する P. C. はないように思える. しかしこれらのデータ種類をほぼ満足する P. C. として, FACOM 230-75 アレイプロセッサ (APU)[8),12)] があり, 高級言語として AP-FORTRAN[9)] を備えている. 実験はこの言語を用い, この計算機によって行うことにした. 一方, 比較の対象としては FACOM 230-75 CPU による科学技術計算ライブラリ SSL II の変換サブルーチンの 1 つである CFTN と PNR サブルーチンを使用した. これらは Singleton algorithm[10)] に基づき, 前者は 8・2 基底に基づき W 表の作成と変換を, 後者はビット反転を行うものである. 一方, 並列算法としては, CFTN に対して並列 Cox 法と Tuttle の算法, PNR に対しては 3.4 節の算法を使用する. ここでは速度と精度をテストの対象にした. 精度の評価には, 乱数データを発生させ, それを変換し, さらに逆変換したものと元の乱数データとの相対誤差の一様ノルムを取ることにした. これらの結果を**図 7** に示している.

図 7 から, この P. C. によって実現した並列 Cox 法と Tuttle の算法は, 例えば複素 2,048 点で, CFTN の 6.6 倍, ビット反転は同点数で 3.3 倍の効率向上となった. また, このビット反転のための並列算法は, 大きさ N の整数型一次元配列を余分に必要とするが, 通常の計算機によって用いられたとしても, Singleton のそれより速いことも測定された. また, 精度に関しては CFTN より悪くなっている. これは CFTN の W 表作成が, 単精度演算で倍精度演算を行う工夫をしているのが一因であると思われる.

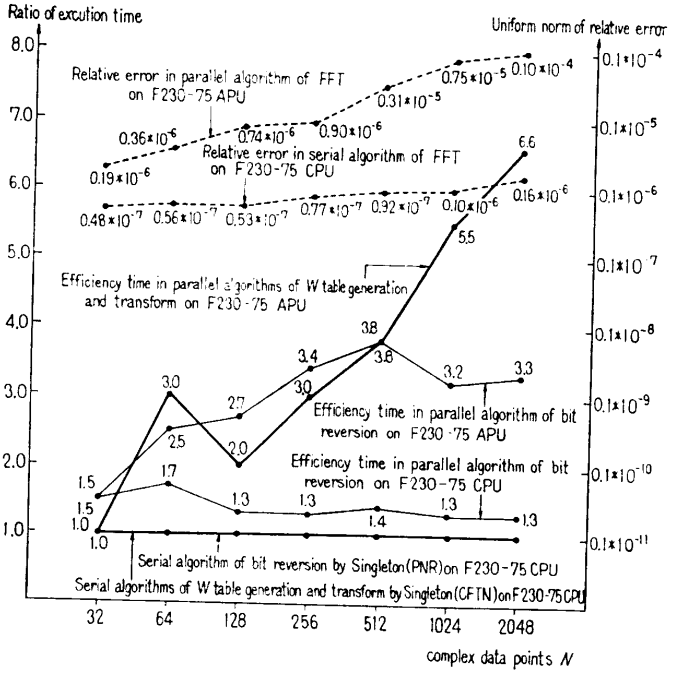

図 7 順次 FFT 算法と並列 FFT 算法との実行時間比と相対誤差
Fig. 7 Excution time ratios and relative errors in serial vs parallel algorithm of FFT.

一方, 3.3 節の算法は BVCT の代りに, VCT 除算と乗算を用いて実現可能ではあるが, 一般に P. C. の VCT 除算は遅いので, 現在のところ実験していない.

6. むすび

本論文では, まず FFT のための新しい並列算法を

説明するために，最初にパイプライン計算機 (P. C.) を想定し，その基本的性質，機能などについて述べ．通常の FFT についてふれたあと，この P. C. によって FFT を行うための並列算法を提案した．そしてこの算法の演算量を評価した(表 1)．さらにこの算法を実際の P. C. により実行し，その処理時間と結果の精度を通度を通常の計算機のそれと比較した(図 7)．図 7は，本論文で述べた P. C. による並列 FFT 算法が，通常の計算機による順次 FFT 算法より，効率良いことを示している．なお使用された並列算法は，サブルーチン化されて実用に供されている．

一方，現在のままでも 2. 1. 2 項で述べた四則演算のほかに，P. C. の性質を生かした複合ベクトル演算があればさらに高速化するし，データ数Nを固定化することによってもさらに高速化する．また，ここで述べられなかった算法に Cooley-Tukey 正順入力正順出力 FFT があるが，これはビット・ベクトルとインダイレクト・ベクトルを持つ P. C. によって容易に実現可能であることを指摘しておく．そしてここで提案した算法はW表作成時に recursive doubling 技法[13]を用いれば，すべてアレイ計算機に適用出来ると思われる．

今後の課題として，演算量をさらに減らすために基底を大きく取った場合の研究，次にここでの並列算法は順次算法よりも作業用領域を必要とするので，これを少なくする研究，そして重要な多次元 FFT の研究がある，これらについては今後の課題である．

最後に本論文作成にあたって，種々お世話になった富士通(株)鈴木滋氏，山下真一郎氏，杉本南海夫氏，鈴木千里氏を始め，LP 部の諸氏に深謝いたします．

参 考 文 献

1) 一松　信：初等関数の数値計算，教育出版株式会社，東京，pp. 129-145 (1974).
2) 内田俊一：信号処理用プロセッサ，情報処理，Vol. 18, No. 4, pp. 402-408 (1977).
3) 加藤満左夫，苗村憲司：並列処理計算機，オーム社，東京，pp. 153-162 (1976).
4) Tuttle, P. G.: Private Communication (1977).
5) Ward, R. C.: The QR Algorithm and Hyman's Method on Vector Computers, Mathematics of Computation, Vol. 30, No. 133, pp. 132-142 (1976).
6) Stone, H. S.: Parallel Tridiagonal Equation Solvers, ACM Transaction on Mathematical Software, Vol. 1, No. 4, pp. 289-307 (1975).
7) Tuttle, P. G.: Implementation of Selected Eigenvalue Algorithm on a Vector Computer, NPGD-TM-330, Babcock & Wilcox (1975).
8) 富士通：FACOM 230-75 アレイプロセッサハードウェア解説書 (1975).
9) 富士通：FACOM MVII AP-FORTRAN 文法書 (1977).
10) Singleton, R. C.: Convolution Procedure Based on The Fast Fourier Transform, Communication of The ACM, Vol. 12, No. 3, pp. 179-184 (1969).
11) 富士通：FACOM FORTRAN SSL II 使用手引書 (1977).
12) 三輪，乾，久米，内田：FACOM 230-75 アレイプロセッサ，情報処理，Vol. 18, No. 4, pp. 410-415 (1977).
13) Stone, H. S.: An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations, JACM, Vol. 20, No. 1, pp. 27-38 (1973).
14) Cooley, J. W. and Tukey, J. W.: An Algorithm for the Machine Calculation of Complex Fourier Series, Mathematics of Computation, Vol. 29, No. 90, pp. 297-301 (1965).
15) Tuttle, P. G.: Private Communication (1979).

付　録

Tuttle の並列変換の優位性を示すために，一応ここでは(10)の Sande-Tukey algorithm に基づき変換を行うための並列算法を示す．P. C. によって変換を行うための問題点は，各 stage において変換を行うために必要とされる X_i の要素を示すアドレスを並列に求めることである．そこでその並列アドレス計算法の1つを図1の例を用いて説明する．各 stage で参照する X_0, X_1, X_2 の要素アドレスを示す大きさ8の整数型一次元配列 AD を用意する．第 1，第 2，第 3 stage の AD の内容を k^1, k^2, k^3 とし，2進3桁で表わすと

$$k^1 = k_2 2^2 + k_1 2^1 + k_0$$
$$k^2 = k_1 2^2 + k_2 2^1 + k_0$$
$$k^3 = k_1 2^2 + k_0 2^1 + k_2$$

となっている．すなわち k^1 は0から7までの数列そのもの，k^2 は k^1 の k_2 と k_1 を反転したもの，k^3 は k^2 の k_2 と k_0 を反転したものから成っている．これを実現するためには BVCT を用いる．こうして AD をビット操作で修正しつつ，AD をインデックスとして，IVCT を用いて，X_0, X_1, X_2 の要素アドレスを参照することによって並列変換を行う．以上から，(10)に基づいた並列変換はアドレス計算が入るため，速度に関する Tuttle の算法の優位性は明らかである（本文 2. 2 節と 3. 2 節を参照）．

(昭和 53 年 10 月 4 日受付)
(昭和 54 年 10 月 25 日採録)