計算機アーキテクチャ　104－8
システムソフトウェアと
オペレーティング・システム　62－8
（１９９４．　１．　２８）

並列ＦＦＴと通信時間の評価

武田利浩*，丹野州宣*，堀口進**

*山形大学工学部電子情報工学科
**北陸先端科学技術大学院大学

本文では８隣接プロセッサ・アレイに適用できる基数４のバタフライ演算を用いた並列FFTアルゴリズムを提案する。本アルゴリズムは基数４のバタフライ演算の性質と８隣接プロセッサ・アレイ上にシャフル行主体順序で配列されたデータの性質を巧みに組み合わせたものである。本アルゴリズムを８隣接プロセッサ・アレイを持つ商用の超並列計算機であるMasPar計算機(MP-1)に実装し、処理時間とその通信時間の関係を調べ、プロセッサ数に対するその性能を評価する。

An Evaluation of Communication Cost for Parallel FFT Algorithms

Toshihiro TAKETA*, Kuninobu TANNO*, Susumu HORIGUCHI**

*Department of Electrical and Information Engineering, Yamagata University
Yonezawa, Yamagata 992, Japan
**Japan Advanced Institute of Science and Technology
15 Asahidai, Tatunokuti, Noumi, Ishikawa 923-12, Japan

In this paper, we describe parallel FFT algorithms on massively parallel processor arrays with eight-neighbor mesh interconnection networks. These algorithms are obtained by combining properties of the radix 4 butterfly computation and shuffled row-major order to index data on processor array. The algorithms are implemented on a commercial massively parallel computer MP-1 and their communication costs are evaluated.

1．まえがき

高速フーリエ変換（FFT）[1]はディジタル信号処理の基本技術としてスペクトル解析、ディジタルフィルタ、音声認識、画像処理などに広く利用されている。FFTを利用する応用分野が広がると同時に、処理されるデータ量もまた増大の一途をたどり、その高速処理が強く望まれている[2]。

FFTを高速に処理するための一つの方法は、FFTに内包されている並列性を巧みに利用することである。そのため、従来から各種の並列アルゴリズムやそれらのプロセッサ・アレイやマルチプロセッサへのインプリメント（実装）が提案されている。

種々の並列アルゴリズムが研究される一方で、超並列計算機が新しい計算サーバとして登場し、FFTなど信号処理への応用が注目されている[3]。超並列計算機の演算処理部は多数の演算要素（PE）とそれらを相互に結合する相互結合網から成る。その相互結合網の形態としては、木構造[4]、正方格子（メッシュ）[5]、超立方体[6]等が採用されている。超並列計算機は、既に商用機が多数存在し、その中の一つにMasPar社のMP-1がある。この超並列計算機は、8個の隣接PE間で通信可能なSIMD型計算機である[7]。

本文ではそのような8隣接プロセッサ・アレイに適用できる基数4のバタフライ演算を用いた並列FFTアルゴリズムを提案する。本アルゴリズムは基数4のバタフライ演算の性質と8隣接プロセッサ・アレイ上にシャフル行主体順序で配列されたデータの性質を巧みに組み合わせたものである。本アルゴリズムをMasPar計算機(MP-1)に実装し、処理時間とその通信時間の関係を調べ、プロセッサ数に対するその性能を評価する。

2．基数4の並列FFTアルゴリズム

N個の複素入力データ列をg(n)、$0 \leqq n \leqq N-1$で表す。$N=4^m$の場合、nを4進m桁で表示すればg(n)は$g(n_{m-1},..,n_1,n_0)$と表すことができる。ただし、$0 \leqq n_i \leqq 3$である。また、g(n)のフーリエ係数をG(n)とすれば、G(n)も$G(n_{m-1},..,n_1,n_0)$と表示することができる。フーリエ係数をG(n)は、$W=\exp(-2\pi j/N)$として、

$$G(n)=\frac{1}{N}\sum_{k=0}^{N-1} g(k)W^{nk} \tag{1}$$

と書ける（以下、1／Nを省略）。また、フーリエ係数$G(n_{m-1},..,n_1,n_0)$は、

$$\begin{aligned} &G(n_{m-1},n_{m-2},\cdots,n_1,n_0) \\ &=\sum_{k_{m-1}=0}^{3}\cdots\sum_{k_1=0}^{3}\sum_{k_0=0}^{3} g(k_{m-1},..,k_1,k_0)W^{n(4^{m-1}k_{(m-1)}+..+4k_1+k_0)} \\ &=\sum_{k_0=0}^{3}\sum_{k_1=0}^{3}\cdots\left\{\sum_{k_{m-1}=0}^{3} g(k_{m-1},\cdots,k_1,k_0)W^{4^{m-1}nk_{m-1}}\right\}W^{4^{m-2}nk_{m-2}}\cdots W^{4nk_1}W^{nk_0} \end{aligned} \tag{2}$$

のように表される[8]。以下の説明において、データ数$N=4^i$に対する回転子を$W_i=\exp(-2\pi j/4^i)$、第i段目、$1 \leqq i \leqq m$、のFFTの出力は$n_{i-1}=0, 1, 2, 3$をパラメータとして$g_i(n_0,n_1,..,n_{i-1},k_{m-i-1},..,k_0)$と表示する。また、$g_0(k_{m-1},..,k_1,k_0)$は初期データ$g(k_{m-1},..,k_1,k_0)$を表す。以下、時間間引き型のFFTについて考察するが、周波数間引き型についても同様に議論できる。

時間間引き型では式（2）において、まずk_{m-1}に対しΣ操作を施す。｛｝内を第1段目として展開すると、次式のような4点のDFTの式が得られる。

$$\begin{aligned} &g_1(n_0,k_{m-2},..., k_1,k_0) \\ &\qquad=\sum_{k_{m-1}=0}^{3} g(k_{m-1},..., k_1,k_0)W_1^{n_0k_{m-1}} \end{aligned} \tag{3}$$

式（3）の右辺のこの形は、基数4の基本演算となっており、第1段目は直接4点のフーリエ変換として与えられる。一般に、k_{m-i}（$2 \leqq i \leqq m$）に対してΣ操作を施す第i段目は次式のようになる。

$$\begin{aligned} &g_i(n_0,n_1,..., n_{i-1},k_{m-i-1},..., k_1,k_0) \\ &=\sum_{k_{m-i}=0}^{3}\{g_{i-1}(n_0,..., n_{i-2},k_{m-i},..., k_1,k_0)W_i^{n_ik_{m-i}}W_{i-1}^{n_{i-1}k_{m-i}}\cdots W_2^{n_2k_{m-i}}\}W_1^{n_1k_{m-i}} \\ &=\sum_{k_{m-i}=0}^{3}\{g_{i-1}(n_0,..., n_{i-2},k_{m-i},..., k_1,k_0)W^{nk_{m-i}}\}W_1^{n_ik_{m-i}} \end{aligned} \tag{4}$$

W_1はデータ数$N=4$に対する回転子であるから、式（4）は、｛｝内のデータに対する4点

のDFTの式に相当する。このとき回転子W^{hkm-i}のhの値は

$$h=(4^{m-i}n_0+4^{m-i+1}n_1+\cdots+4^{m-2}n_{i-2}) \quad (5)$$

より求まる。なお、式（3）からわかるようにi＝1のとき、hは0である。以上のようにして、第i段目の出力と回転子W^{hkm-i}のhの値が得られる。第m段目（最終段）は、$n_{m-1}=0$から順番に演算結果を戻す事になり、この高速フーリエ変換の結果は$G(n_0,n_1,\ldots,n_{m-2},n_{m-1})$のように、元の$G(n_{m-1},\ldots,n_1,n_0)$の桁逆順で求まる。

ところで、式（3）、（4）は基数4のバタフライ演算と呼ばれる次のような加減算

$$\begin{aligned}
g_i(0)&= g_{i-1}(0)v_0 + g_{i-1}(1)v_1 + g_{i-1}(2)v_2 + g_{i-1}(3)v_3\\
g_i(1)&= g_{i-1}(0)v_0 - j\,g_{i-1}(1)v_1 - g_{i-1}(2)v_2 + j\,g_{i-1}(3)v_3\\
g_i(2)&= g_{i-1}(0)v_0 - g_{i-1}(1)v_1 + g_{i-1}(2)v_2 - g_{i-1}(3)v_3\\
g_i(3)&= g_{i-1}(0)v_0 + j\,g_{i-1}(1)v_1 - g_{i-1}(2)v_2 - j\,g_{i-1}(3)v_3
\end{aligned} \quad (6)$$

を繰り返すことにより容易に求められる[17]。ここで、$g_i(n_{i-1})=g_i(n_0,n_1,\ldots,n_{i-1},k_{m-i-1},\ldots,k_1,k_0)$で、$n_{i-1}=0$，1，2，3である。式（4）より、$v_i=W^{h\cdot i}$、i＝0，1，2，3である。ところで、$v_0$は常に1であるので、式（6）は3回の乗算と8回の加算で求まる。

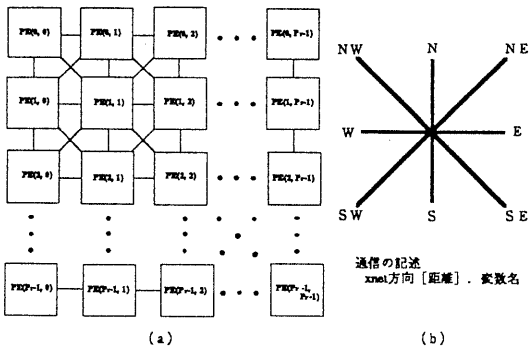

図1 8隣接プロセッサ・アレイの構成

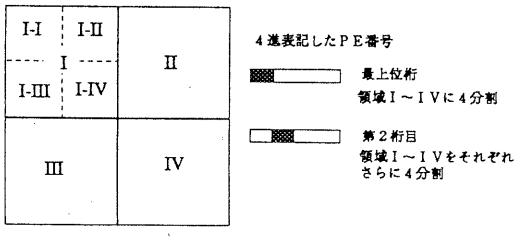

図2 シャフル行主体順序の性質

本文では、式（6）で与えられるバタフライ演算を8隣接プロセッサ・アレイ上で計算するするインプレス型と非インプレス型と呼ばれる2種類の演算操作を定義し、それらを繰り返し実行することにより、高速な並列FFTを行なう。

3． 8隣接プロセッサ・アレイ

3．1 8隣接プロセッサ・アレイの構成

$P=P_r\times P_r$個のプロセッサから成る8隣接プロセッサ・アレイの構成を図1（a）に示す。ここで、$P_r=2^q$であり、x，y＝1，2，・・・，P_r-1をそれぞれ行と列の番号として各プロセッサをPE(x;y)と表記する。あるいは、プロセッサ・アレイをシャフル行主体順序で番号図付けられているものとして、z＝0，1，・・・，P－1とし、一連番号でPE(z)と表す。図示のように各プロセッサは3、5あるいは8本のリンクを持ち、このリンクを介して隣接プロセッサと通信する。通信は、図1（b）のように、8方向（N,NE,E,SE,S,SW,W,NW）の方向と、PE間の相対距離によって指定される。ここで、通信は半二重通信を仮定し、各PEにおいて、転送要求を持つリンクがすべて同時に並列に転送する方法を同時転送法（全ポート通信[9]）、各リンクが順々に連続して転送する方法を順次転送法（1ポート通信）と呼ぶ。また、各PEはFFTの計算に必要なデータを格納するための局所メモリを備えているものとする。

3．2 シャフル行主体順序

8隣接プロセッサ・アレイ上でのFFTは、N個の入力データ列の初期割り付けは、シャフル行主体順序に並べられる。シャフル行主体順序[10]で並べられたデータは次のような性質を持っている。

図1のプロセッサ・アレイ上にN個のデータ$g(n_{m-1},\ldots,n_1,n_0)$をシャフル行主体順序で並べ、アレイを図2のように4分割するとN個のデータは領域I：$g(0,n_{m-2},\ldots,n_1,n_0)$、領域 II：$g(1,n_m$

$_{-2}$,..,n_1,n_0)、領域III：g(2,n_{m-2},..,n_1,n_0)、領域IV：g(3,n_{m-2},..,n_1,n_0)のように番号の小さい順に各領域に4分割されて配置される。すなわち、番号の最上位桁0，1，2，3により各領域のデータは区別される。さらに、領域Iを4分割すると領域 I-I：g(0,0,n_{m-3},..,n_1,n_0)、領域I-II：g(0,1,n_{m-3},..,n_1,n_0)、領域I-III：g(0,2,n_{m-3},..,n_1,n_0)、領域I-IV：g(0,3,n_{m-3},..,n_1,n_0)と配置され、この場合は上位1桁目と2桁目の値により4分割された領域が区別される。他の領域についても同様であり、さらに分割を続けても同様の再帰性が保持される。本アルゴリズムはこのシャフル行主体順序の持つ性質を巧みに利用する。以下、再帰的な分割で使われた桁、すなわち、k回目の分割であれば、k番目の桁が0である領域、つまり領域I、領域I−I、領域II−Iなどの各（サブ）領域を再帰的に第1領域と呼ぶ。同様に1であれば第2領域、2であれば第3領域、3であれば第4領域と呼ぶ。

3.3　バタフライ演算

8隣接プロセッサ・アレイ上で式（6）で与えられる基数4のバタフライ演算を処理するためのインプレス型と非インプレス型演算を定義する。

3.3.1　インプレス型演算

図3（a）に示すように8隣接プロセッサ・アレイ上のPE(x;y)とこのPEからdだけ離れた3つのPE(x;y+d)、PE(x+d;y)、PE(x+d;y+d)を考える。ここで、$d=2^{q-i}$、i＝1、2、..、qである。また、各PEは図示のようにそれぞれデータ

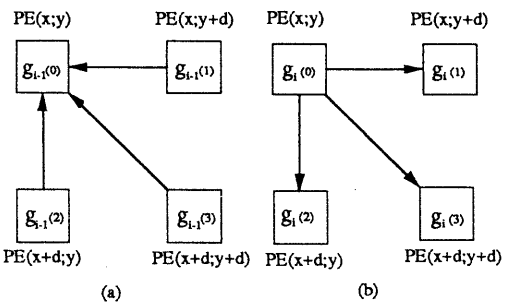

図3 インプレイス演算

（値）$g_{i-1}(0)$、$g_{i-1}(1)$、$g_{i-1}(2)$、$g_{i-1}(3)$を格納しているものとする。

1）PE(x;y+d)，PE(x+d;y)，PE(x+d,y+d)はそれぞれのデータ$g_{i-1}(1)$、$g_{i-1}(2)$、$g_{i-1}(3)$をPE(x;y)に送出する。

2）PE(x;y)は$g_{i-1}(0)$、$g_{i-1}(1)$、$g_{i-1}(2)$、$g_{i-1}(3)$に対して式（6）で与えられる4点DFTを行う。

3）PE(x;y)はバタフライ演算の結果$g_i(0)$、$g_i(1)$、$g_i(2)$、$g_i(3)$を図3（b）に示すようにインプレスに返すものとする。

1）、2）、3）で述べた一連の操作を8隣接プロセッサ・アレイ上での式（6）で与えられる基数4のバタフライ演算に対するインプレス型演算と定義する。

3.3.2　非インプレス型演算

イプレス型演算と同様にPE(x;y)を考えるが、非イプレス型演算においてはPE(x;y)がデータ$g_{i-1}(0)$、$g_{i-1}(1)$、$g_{i-1}(2)$、$g_{i-1}(3)$を格納する（図4（a）参照）。

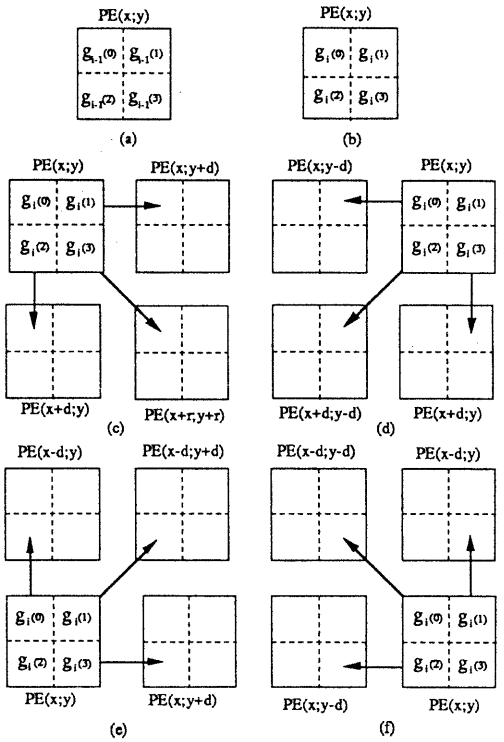

図4 非インプレイス演算

1）PE(x;y)は$g_{i-1}(0)$、$g_{i-1}(1)$、$g_{i-1}(2)$、$g_{i-1}(3)$に対して式（6）で与えられる4点DFTを行い、結果$g_i(0)$、$g_i(1)$、$g_i(2)$、$g_i(3)$を得る（図4（b）参照）。

2）PE(x,y)は結果$g_i(0)$、$g_i(1)$、$g_i(2)$、$g_i(3)$を図4（c）から（f）に示すようにd離れた3つのPEに転送する。その転送方向はPE(x,y)が図2で説明された領域のどこに位置するかにより異なる。第1領域なら図4（c）、第2領域なら同図（d）、第3領域なら同図（e）、第4領域なら同図（f）のようになる。

1）、2）で述べた一連の操作を式（6）で与えられる基数4のバタフライ演算に対する非インプレス型演算と定義する。

アルゴリズム1

```
// データのマッピング
PE(z).data=g(z)
for (k=1,d=Pr/2; k<=m; k++, d/=2) {
  // 第1領域？: PE(z_{q-1},...,z_{q-k}(=0),..,z_0)
  if (chk4(k,z)==0) // 第1領域
    // インプレイス演算
    // Get Data
      // g_{k-1}(n_0,...,k_{q-k}(=0),k_{q-k-1},..,k_0)
    pg(0)=data;
      // g_{k-1}(n_0,...,k_{q-k}(=1),k_{q-k-1},..,k_0)
    pg(1)=xnetE[d].data;
      // g_{k-1}(n_0,...,k_{q-k}(=2),k_{q-k-1},..,k_0)
    pg(2)=xnetS[d].data;
      // g_{k-1}(n_0,...,k_{q-k}(=3),k_{q-k-1},..,k_0)
    pg(3)=xnetSE[d].data;
    // 4点DFT
    Radix4Butterfly(k, pg(0), pg(1), pg(2), pg(3));
    // Put Data
      // g_k(n_0,...,n_{k-1}(=0),k_{q-k-1},..,k_0)
    data=pg(0);
      // g_k(n_0,...,n_{k-1}(=1),k_{q-k-1},..,k_0)
    xnetE[d].data=pg(1);
      // g_k(n_0,...,n_{k-1}(=2),k_{q-k-1},..,k_0)
    xnetS[d].data=pg(2);
      // g_k(n_0,...,n_{k-1}(=3),k_{q-k-1},..,k_0)
    xnetSE[d].data=pg(3);
}
```

4．並列FFTアルゴリズム

8隣接プロセッサ・アレイ上での基数4の並列1次元FFTアルゴリズムの実装について考察するが、N≦PとN>Pの場合に分けて考える。プロセッサとデータの番号付けは、4進表記を用いて行うと都合がよい。N個の入力データを$g(n_{m-1},n_{m-2},..,n_1,n_0)$、i段目のバタフライ演算の結果を$g(n_0,n_1,..,n_{i-1},k_{m-i-1},..,k_1,k_0)$と表す。また、P個のPEは2次元アレイとしての表記であるPE(x;y)だけでなく、シャフル行主体順序で番号づけされているものと仮定し、$PE(z_{q-1},z_{q-2},..,z_1,z_0)$とも表わす。

アルゴリズム中において、他のPEに割り付けられたデータの参照すなわち通信は相対的な距離と方向の指定と変数名を用いて記述するものとする。方向には、先の図1（b）で示した8方向（N,...)を指定することができる。あるPEが、N方向に距離dだけ離れたPEの変数dataをアクセスするには、xnetN[d].dataの様に表記する。

4．1　N≦Pの場合

この場合は、先に定義したインプレイス型演算を繰り返すことによって、FFT演算が実現される。すなわち、第1段目は、領域を4分割し、第1領域のPEが、距離Pr/2で、インプレイス型演算を行う。第2段目では、第1段で4分割された領域それぞれで、領域を4分割し、距離を半分にして、第1段と同様にインプレイス型演算を行う。以下、第m段目まで、再帰的に領域を4分割、距離を半分にして、インプレイス型演算を繰り返すことで、m段のバタフライ演算が完成し、FFTが終了する。

次ぎに、MP-1上へのアルゴリズムの実装を考える。各PEに$g(n_{m-1},..,n_1,n_0)$を格納する変数dataとバタフライ演算で使用する配列pg[4]を用意する。インプレイス型演算は、データの収集と基数4のバタフライ演算（4点DFT演算）とデータの返送に分解して記述し、特に4点DFT演算のために、関数Radix4Butterfly()を用意する。さらに、4進数nのk番目の桁を返す関数chk4(k,n)を定義する。これらの関数を用いて、実装アルゴリズムを書くと、アルゴリズム1の

ようになる。

4．2　N＞Pの場合

N＞Pの場合は１個のPEに４の冪乗個のデータを割り付けることにより基数４の並列FFTを実現する。データの割付法としては相似割付法と重畳割付法と呼ぶ２通りの方法を考える。N≦Pの場合と同様にデータは$g(n_{m-1},n_{m-2},..,n_1,n_0)$で、PEはシャフル行主体順序で配列されているものと仮定し、$PE(z_{q-1},z_{q-2},..,z_1,z_0)$で表す。ただし、m≧q＋１である。

２つのデータ割付法を説明するが、データは$N_r \times N_r$の２次元配列にシャフル行主体順序で並んでいるものとする。２つのデータ割付法は、それぞれ、図（a）と（b）のようになる。相似割付法はこのデータ$g(n_{m-1},n_{m-2},..,n_1,n_0)$のnの上位q桁をＰＥの指定に、下位m－q桁をＰＥ内での２次元配列の位置の指定に使う。すなわち、データ$g(n_{m-1},n_{m-2},..,n_1,n_0)$を$PE(n_{m-1},n_{m-2},..,n_{m-q})$に割り付ける方法である。一方、重畳割付法はnの上位m－q桁をＰＥ内での２次元配列の位置に、下位q桁をＰＥの指定の指定に使う。すなわち、データ$g(n_{m-1},n_{m-2},..,n_1,n_0)$を$PE(n_{q-1},n_{q-2},..,n_1,n_0)$に割り付ける方法である。

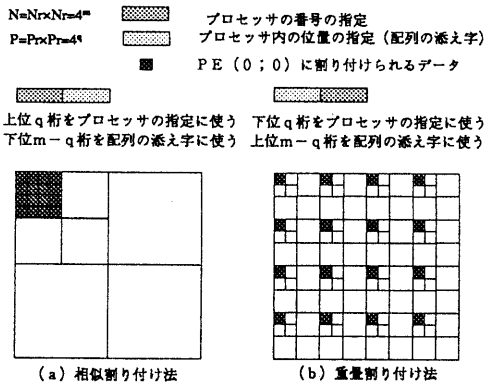

図５ プロセッサ・アレイへの割付法

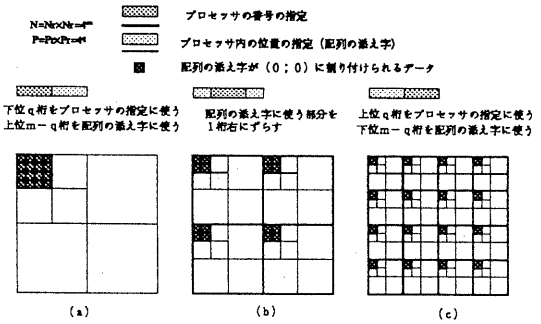

図 6 重畳割付法のデータの割付位置の変化

4．2．1　相似割付法

相似割付法の場合は、第１～ｑ段目までは、N≦Pの場合と同様に、インプレイス型演算を繰返し、残りのｑ＋１～m段は、必要なデータが既にＰＥ内にそろっているので、通信無しでm－ｑ段のバタフライ演算を行う。

実装アルゴリズムは、アルゴリズム１をそのまま拡張出来るので、割愛する。

4．2．2　重畳割付法

重畳割付法の場合は、第１～ｑ段目までは、非インプレイス型演算を繰返し、残りのｑ＋１～m段は、必要なデータが既にＰＥ内にそろっているので、通信無しでm－ｑ段のバタフライ演算を行う。重畳割付法では、データの割付が１段毎に変化するのが特徴となっており、m=3,q=2の場合の割り付けの様子を図に示す。本アルゴリズムでは、４点ＤＦＴに必要なデータが同一のＰＥに既に割り付けられているので、ＰＥ内のデータで４点ＤＦＴを行える。次ぎに、データの転送を行うが、転送先は次回の４点ＤＦＴで必要なデータが同一ＰＥ内に揃うように、選ばれる。

このように、本アルゴリズムでは、第１～q段まではバタフライ演算とデータ転送を１段毎に繰り返すが、各PEが4^{m-q}のデータを格納しているので（m－q）段バタフライ演算を行ない、その結果$g_i(n_0,n_1,..,n_{m-q-1},k_{q-1},..,k_1,k_0)$を$PE(n_0,..,n_{n-q-1},k_{2q-m-1},..,k_0)$に転送するようなアルゴリズムも考えられる。しかし、この場合は転送先のPEの数が4^{m-q}個となるため、転送経路が複雑になり８隣接プロセッサ・アレイの通信能力を十分に発揮させることができない。

実装に際しては、非インプレース型演算なので、各ＰＥに入力用配列data(x';y')と出力用配列data'(x";y")を用意する。アルゴリズム１で定義

アルゴリズム２

```
// データのマッピング
penum=lk4(q-m,z);    // 上位ｑ桁
subscript=(mk4(q,z);  // 下位ｍ－ｑ桁
subscript=shuffle(subscript);
PE(penum).data(subscript)=g(z)
Sr=sqrt(Pr/Nr); // dataのサイズはSrxSr
// 基数４ １次元ＦＦＴ
for (k=1,d=Pr/2;k<=q;k++, d/=2) {
  for (sd=Sr/2,y'=0;y'<Sr;y'++)
  for (x'=0;x'<Sr;x'++)
      Radix4Butterfly(k, data(x';y'), data(x';y'+sd),
            data(x'+sd;y'), data(x'+sd;y'+sd));
      // Put Data
      x''=x'<<1 + chkbit(k,x) ;
      y''=y'<<1 + chkbit(k,y);
      switch (chk4(k,z)) {
      case 0: // 第１領域
        data'(x'';y'')=data(x';y');
        xnetE[d].data'(x'';y'')=data(x';y'+sd);
        xnetS[d].data'(x'';y'')=data(x'+sd;y');
        xnetSE[d].data'(x'';y'')=data(x'+sd;y'+sd);
      case 1: // 第２領域
        xnetW.data'(x'';y'')=data(x';y');
        data'(x'';y'')=data(x';y'+sd);
        xnetSW[d].data'(x'';y'')=data(x'+sd;y');
        xnetS[d].data'(x'';y'')=data(x'+sd;y'+sd);
      case 2: // 第３領域
        xnetN[d].data'(x'';y'')=data(x';y');
        xnetNE[d].data'(x'';y'')=data(x';y'+sd);
        data'(x'';y'')=data(x'+sd;y');
        xnetE[d].data'(x'';y'')=data(x'+sd;y'+sd);
      case 3: // 第４領域
        xnetNW[d].data'(x'';y'')=data(x';y');
        xnetN[d].data'(x'';y'')=data(x';y'+sd);
        xnetW[d].data'(x'';y'')=data(x'+sd;y');
        data'(x'';y'')=data(x'+sd;y'+sd);
      }
    }
    swap(data,data');// 入力配列と出力配列の交換
}
// ＰＥ内で実行出来る残りのバタフライ演算
for (k=q+1,sd=Sr/2; k<=m; k++, sd/=2) {
  for (l=0;l<=2^(m-q-k);l++)
    for (h=0;h<=2^(m-q-k);h++)
      x'=insbit(k-q,l,0); y'=insbit(k-q,h,0);
      Radix4Butterfly(k, data(x';y'), data(x';y'+sd),
            data(x'+sd;y'), data(x'+sd;y'+sd));
}
```

した関数の他に、iのkビット目に１ビットjを挿入する関数insbit(k,i,j)、４進数nの上位k桁を返す関数mk4(n,k)と下位k桁を返すlk4(n,k)を定義する。また、配列の名前(ポインタ)を付け変えるswap(ptr1,ptr2)を導入する。各ＰＥにg(n)を格納する２次元配列data(x';y')を用意する。

これらの関数を用いて実装アルゴリズムを記述すると、アルゴリズム２のようになる。

５．　実装アルゴリズムの処理時間

アルゴリズム２をMP-1上に実装し、その処理時間を実測する。MP-1では、全ポート通信は提供していないので、１ポート通信によって、データ転送を行っている。その処理時間を通信、基数４のバタフライ演算、その他の３つのカテゴリーに分け、比較する。

図７に、データ数を４０９６とし、プロセッサ数を、４、１６、６４、２５６、１０２４と

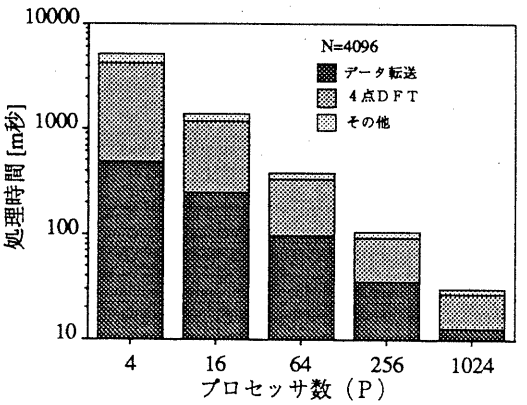

図７ 処理時間の実測値

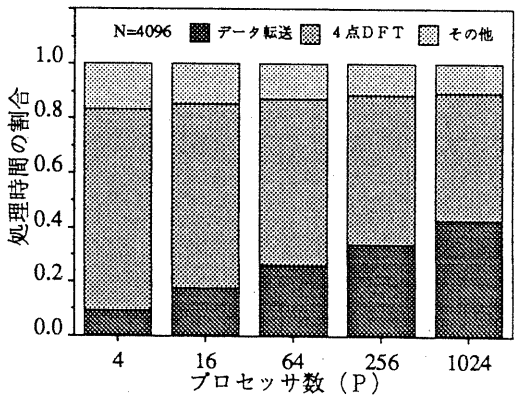

図８ 処理時間の実測値

変化させた時の、処理時間を示す。図示のように、プロセッサ数を4倍する毎に、5116.8、1380.2、375.6、103.9、29.8m秒と、処理速度が1、3．7、13．6、49．2、172．0倍になるような高速化が得られることが分かる。

また、図7を、処理時間全体を1として正規化して書き直すと図8のようになる。図示のように、データ数を固定した場合は、プロセッサ数を増やすに連れて、通信時間の割合が、9．3%、17．8%、25．9%、34．0%、42．5%と増加し、逆に、4点DFT演算の占める割合は、73．9%、67．4%、61．0%、54．3%、46．8%と減少している事がわかる。これは、4点DFT演算は、全てのPEで均等に並列実行されるため、台数効果がそのまま得られるのに対して、プロセッサ数を4倍する毎に、プロセッサ・アレイ上のPE間の最大距離が倍になり、通信を並列に行うことによる台数効果を相殺してしまうためである。ただし、転送時間の絶対値は、475.2、245.9、97.4、35.4、12.7m秒と減少している。したがって、並列度を増すに連れて、データ転送のアルゴリズムの優劣が台数効果を維持するための重要な要因となる。

そのためには、全ポート通信（3ポート通信）、あるいは分割再構成可能な8隣接トーラス構造を提供する計算機を使うことで、転送時間を大幅に短縮することが期待できる。

6．おわりに

本文は基数4のFFTを8隣接プロセッサ・アレイで並列処理するためのアルゴリズムについて報告した。N≦PとN>Pの場合についてアルゴリズムはを考察し、N>Pの場合は2つのデータ割付法とそれらに対するアルゴリズムを提案した。これらのアルゴリズムは基数4のバタフライ演算の性質とシャフル行主体順序の性質を組み合わせることにより実現された。次いで、アルゴリズムをMP-1に実装し、処理時間と通信時間の実測値を示した。その結果、データ数（4096）を固定してプロセッサ数を4、16、64、256、1024と大きくしたとき、処理時間全体に占める通信時間の割合は増加するが、通信時間そのものは大きく減少し、処理速度も1、3．7、13．6、49．2、172．0倍になるような高速化が得られることが分かった。

今後、本アルゴリズムの多次元FFTへの拡張や、分割再構成可能な8隣接トーラス構造を持つ計算機への適用を考慮したい。

謝辞

MasPar（MP-1）を使用するにあたり、ご便宜を頂きました東京大学生産技術研究所喜連川優助教授ならびに（株）理経コンピュータシステム営業部長出雲純氏にお礼申し上げます。

（文献）

(1)J.W.Cooley and J.W.Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Math.Comput., Vol.19, pp.297-301(1965).
(2)川又、樋口、”多次元ディジタル信号処理の基礎（VI・完）”、電情通学誌、Vol.74, No. 10, pp. 1098-1108(1991).
(3)馬場敬信、”超並列マシンへの道”、情報処理、Vol.32, No. 4, pp.348-364 (1991).
(4)山田実、”CM-5”、並列処理シンポジュウムJSPP'92論文集、pp.39-43(1992).
(5)"PARAGAN Supercomputers," Intel Corporation, No. 203/6/92/10k/GA(1992).
(6)"NCUBE Supercomputers," NCUBE Corporation, No. G2-0100(1989).
(7)"MasPar Parallel Application Language (MPL) Reference Manual," MasPar Computer Corporation, Document No. 9302-0000(1991).
(8)伊達玄（訳）／A. V. Oppenhein and R. W. Schafer著、”ディジタル信号処理”、コロナ社(1978).
(9)梅尾博司、”超並列計算機アーキテクチャとそのアルゴリズム”、共立出版(1991).
(10)S.G.Akl, "Parallel Sorting Algorithms," Academic Press, Inc.(1985).