ハイパフォーマンス
コンピューティング 68－7
（1997．10．17）

# 分散共有メモリ型並列計算機上での並列FFTアルゴリズムの実装評価

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

従来から高速フーリエ変換(FFT)の並列アルゴリズムの提案やそれらのプロセッサ・アレイやマルチプロセッサへの実装が報告されている．しかし，それらの多くは，高速な局所通信を利用したものであり，通信アーキテクチャに強く依存し，汎用性には欠けていた．

本稿では，汎用の通信ライブラリであるMPIを利用した並列FFTアルゴリズムをCray T3E上に実装し，その通信時間を評価する．まず，使用する並列計算機Cray T3Eの構成について述べる．ついで２つの並列FFTアルゴリズムStage-by-stage法とMulti-stage法を示し，それらをCray T3E上にMPIを用いて実装し，評価する．

# Implementation of Parallel FFT Algorithms to Massively Parallel Computer with Distributed shared memory and Its Evaluation

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 order to efficiently compute Fast Fourier transform (FFT) various parallel algorithms and their implementation to multiprocessors and multicomputers have been developed. In general, the local interconnection network is more high speed than a global one, but its capability depends on network architecture. On the other hand, the global interconnection network is not so high speed, but it does not depends on network architecture. It provides a flexible communication interface to the programmer.

In this paper, we discuss parallel FFT algorithms on a multiprocessor system with a global interconnection network. We propose two algorithms a stage-by-stage method and a multi-stage method. We implement these algorithms on a Cray T3E parallel computer and measure these communication time.

## 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．その応用分野の広さと処理されるデータ量の大きさから，FFTの高速処理に対する要求は強い[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し，超並列計算機で並列処理することである[3]．従来から各種の並列アルゴリズムの提案やそれらのプロセッサ・アレイやマルチプロセッサへの実装が報告されている[4,5,6]．しかし，これらのアルゴリズムの大部分は基数2のバタフライ演算を用いている．

著者らは，すでに８隣接格子型プロセッサ・アレイ上で局所通信を用いた基数４の並列FFTアルゴリズムの提案とその評価を行なっている[7]．そのアルゴリズムは，本文で示すStage-by-stage法(FFTを一段行う毎にデータの再配置を行う方法)を基数4の場合について8隣接格子網に適応したものにあたる．このアルゴリズムは8隣接格子型プロセッサ・アレイの高速な局所通信ネットワークの性能を十分に引き出すように設計されており，プロセッサの結合形態に密接に関連しているため，他の通信アーキテクチャでは実行ができない．

局所通信ネットワークを用いた通信では，隣接する(直接に通信リンクを持つ)PE間の通信を繰り返す事で目的のPEへデータを渡すため，通信時間は通信するPE間の距離(ホップ数)に依存する．また，局所通信ネットワークの通信の場合，一回の通信で送れるのは１個のatomicデータである．したがって，一回の通信毎にかかるスタートアップ時間(回線の接続待ち時間など)を $t_{lu}$，隣接するPE間の通信時間を $t_{lc}$ とし，距離を $d$ とすれば，その通信時間は $t_{lu}+t_{lc}\times d$ となる．通信時間を小さくするためには，データを移動する距離の総和と通信回数を小さくする事が求められる．送信されるのはatomicデータであるため，通信回数を減らすためには，交換するデータの総量を減らせばよく，両者は矛盾しない．

一方，大域通信ネットワークを用いた通信は，１回の通信で一かたまりのバッファを送ることができ，その通信時間は，距離よりもむしろデータサイズに依存する．ここで，スタートアップ時間を $t_{gu}$，１つのatomicデータの通信時間を $t_{gc}$，データサイズを $S_{data}$ とすれば，$t_{gu}+t_{gc}\times S_{data}$ と書ける．したがって，データの総量を小さくすることが，通信回数を減らすことにはつながらない．

高速な局所通信ネットワークを利用したアルゴリズムに比べ，大域通信ネットワークを利用したアルゴリズムは，通信アーキテクチャによってアルゴリズムを変更する必要が無いため，汎用性を有する．商用機においては，プロセッサの結合形態に強く依存する局所通信ネットワークと共に大域通信ネットワークを提供するものや大域通信ネットワークのみを提供するものもある[8,9,10]．さらに，通信アーキテクチャに依存しないメッセージ送信ライブラリであるMPI(Message Passing Interface)[11]の提案と規格化が進められている．したがって，大域通信ネットワークを用いた並列FFTアルゴリズムについても，十分に検討する必要がある．

本文では，大域通信を利用した基数Rの並列FFTアルゴリズムとその通信コストについて述べ，商用の超並列計算機への実装にも触れる．まず，通信をプロセッサの結合形態に依存しない形で抽象化したときに導き出される2つの並列FFTアルゴリズム，Stage-by-stage法とMulti-stage法を示す．ついで，基数Rに一般化した並列FFTアルゴリズムの通信コストを見積もり，データ交換の戦略(Stage-by-stage法とMulti-stage法)と転送モード(atomic型とbuffer型)の影響を検討する．また，本アルゴリズムをCray T3E上に実装し，通信時間を評価する．最後に，全体をまとめる．

## 2．並列FFTアルゴリズム

### 2．1．　データ割付法

入力データ数をN=$R^m$，PE数をP=$R^q$とすると，入力データ列およびPEは，それぞれR進表記で$g(n_{m-1},n_{m-2},...,n_1,n_0)$，$PE(n_{q-1},n_{q-2},...,n_0)$とかける．1個のPEにRの冪乗個のデータを割り付けることにより基数R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通りの方法が考えられる．

相似割付法は，入力データ列をq回再帰的にR

回分割し，各領域の添字に対応した添字をもつPEにその領域に含まれるデータを格納する方法である．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(m-q)回再帰的にR分割し，下位q桁$n_{q-1},n_{q-2},...,n_1,n_0$が等しいデータを同じ番号を持つPEに重ねて格納する方法である．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．

2つのデータ割付法を説明したが，アルゴリズムの並列度を考慮すると重畳割付法が相似割付法より優れているので，以下，重畳割付法を用いるものとする．

## 2．2．　Stage-by-stage法

各PEが$R^{m-q}$個のデータを格納しているのでデータの再配置なしに(m－q)段のバタフライ演算を行えるが，本方法は第１段～第q段までは１段毎のバタフライ演算とデータの転送を繰り返す．残りの(m-q)段はデータの再配置なしに行える．アルゴリズムは図１のようになる．

```
// Algorithm 1 Stage-by-stage法
SuperpositionMapping();  // 重畳割付
// 1～q段: q段
for (i=1; i<=q; i++) {
        Butterfly( i, 1);     // バタフライ演算
        ReMapping( i, 1);  // データの再配置
}
// q+1～m段: m-q段
for ( k=1; i<=m; i++, k++) {
        Butterfly( i, k);
}
```

図１ アルゴリズム１：Stage-by-stage法

ここで，SuperpositionMapping()は，入力データ列を重畳割付法を用いて各PEに割付ける関数である．Butterfly( i, j)は，各PEの配列の添え字のR進j桁をパラメータとしてi段目のバタフライ演算を行う関数である．ReMapping( i, j)は，データの再割付けを行う関数であり，iは終了したバタフライ演算の段数，jは前回の再配置から行ったバタフライ演算の段数である．

## 2．3．　Multi-stages 法

各PEが$R^{m-q}$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$PE(n_0,...,n_{n-q-1},k_{q-m-1},...,k_0)$に転送するようなアルゴリズムも考えられる．アルゴリズムは図２のようになる．ここで，ceil()は，シーリング関数である．

```
// Algorithm 2 Multi-stage法
SuperpositionMapping();
// 1～(m-q)c段:(m-q)c段
for (j=0, c=ceil(q/(m-q)); j<c; j++) {
        for (k=1; k<=(m-q); k++) {
                Butterfly( j*c+k, k);
        }
        ReMapping( j, (m-q));
}
// (m-q)c+1～m段:  (m-(m-q)c)段, 最大(m-q)段
for (i=(mq)*c+1, k=m-(m-q)*c; i<=m; i++, k++)
{
        Butterfly( i, k);
}
```

図２ アルゴリズム２：Multi-stage法

## 3．通信時間の評価

ここでは，基数Rに一般化した並列FFT処理に要する演算および通信時間を見積もる．データ交換の戦略(Stage-by-stage法とMulti-stage法)と転送モード(atomic型とbuffer型)の影響を検討する．

入力データ数を$N=R^m$，プロセッサ数を$P=R^q$とする．FFTを完了するには，$R^{m-q-1}$個の基数Rのバタフライ演算をm段行う必要があり，本アルゴリズムでは演算をP個のプロセッサに均等に割り付けるから，一つの基数Rのバタフライ演算時間を$t_{Rb}$とすると，FFTが完了するまでのバタフライ演算時間の合計は，次式で表せる．

$$T_{butterfly}(m,q) = R^{m-q-1} \times m \times t_{Rb} \qquad (1)$$

基数Rのバタフライ演算時間の合計は，データ交換の戦略，データ転送のモードによらず，データ数を定めるmと，プロセッサ数を定めるqにのみ依存する．したがって，データ交換の戦略，データ転送のモードの与える影響を調べるには，通信時間のみを考慮すればよい．以下，基数RのFFT処理に要する通信時間を見積もる．

大域通信ネットワークを用いた１回の通信時

間は，$t_{gu}+t_{gc}\times S_{data}$であった．FFTに要する通信時間の合計は，総通信回数と交換されるデータの総量によって定まるので，通信回数の合計と転送されるデータの総量を求めればよい．

初期割り付けあるいは前回の転送からデータの交換無しにL段のFFTを行ったとすると，次段FFTで任意のPEの保持するデータを必要とするPEの数は，次式で表せる．

$$N_{addr}(L)=R^{L} \tag{2}$$

このうち同一のPEに残されるデータを除いた，他のPEへの転送が実際に必要なデータの送り先PEの数は，次式で与えられる．

$$N_{dest}(L)=R^{L}-1 \tag{3}$$

さらに，他のPEへ転送されるべきデータ数は，次式のように得られる．

$$N_{data}(L)=R^{m-q}\times\frac{N_{dest}(L)}{N_{data}(L)}=\frac{N(R^{L}-1)}{P\times R^{L}} \tag{4}$$

入力データ列g(n)は，複素数であったから，1つのデータは2つのatomicデータから構成される．buffer型の転送を行うなら1回の通信で，atomic型の転送を行うなら2回の通信が必要であるので，通信回数の合計は，次のようになる．

$$N_{trans}(L)=\begin{cases}N_{data}(L)\times 2 & (atomic)\\ N_{dest}(L) & (buffer)\end{cases} \tag{5}$$

また，転送するデータサイズは次式で与えられる．

$$S_{data}(L)=N_{data}(L)\times 2 \tag{6}$$

したがって，FFTに要する通信の回数は，次式で得られる．

$$N_{trans}FFT(m,q)=\begin{cases}N_{trans}(1)\times q & (stage-by-stage)\\ N_{trans}(z)+N_{trans}(m-q)\times c & (multi-stage)\end{cases} \tag{7}$$

転送するデータのサイズは次式のようになる．

$$S_{data}FFT(m,q)=\begin{cases}S_{data}(1)\times q & (stage-by-stage)\\ S_{data}(z)+S_{data}(m-q)\times c & (multi-stage)\end{cases} \tag{8}$$

最後にFFT全体の通信時間は次式で得られる．

$$T_{trans}FFT(m,q)=t_{gu}\times N_{trans}FFT(m,q)+t_{gc}\times S_{data}FFT(m,q) \tag{9}$$

これで基数Rの並列FFTアルゴリズムの実行に要する通信時間を見積もる式が得られた．以下，式(2)-(9)を用いて通信コストを評価する．

基数R=4，データ数N=4096(=$4^6$)とした場合のPE1つあたりの通信回数と交換するデータの総量をそれぞれ図3と図4に示す．図3より通信回数はデータ交換の戦略と転送モードに強く依

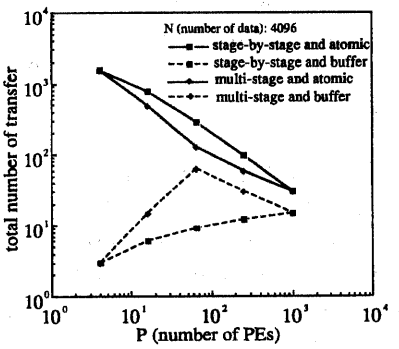


図3　総通信回数

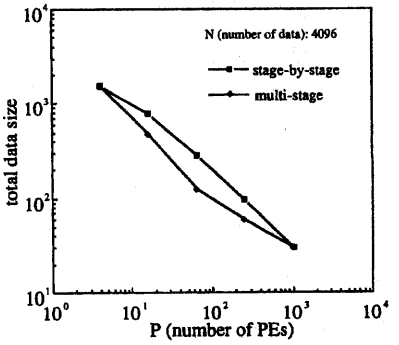


図4　送信を要するデータの総量

存し，大きく違う事が分かる．atomic型の場合は,Stage-by-stage法とMulti-stage法のいずれもPE数の増加とともに通信回数が減少している．これは，PE数の増加は1つのPEに割り当てられるデータ数の減少をもたらすためであり，図4のデータの総量からもこのことが読み取れる．また，buffer型の場合はStage-by-stage法のとき，緩やかに増加している．Stage-by-stage法では，バタフライ演算を1段毎にデータを交換するのでその際の通信回数は式(3)より3回となる．ここで，アルゴリズム1を見れば明らかなように，データの交換は1～q段まで行われるから，qが大きいつまりPE数が大きい程通信回数は増えるこ

とになる．Multi-stage法の場合は，PE数が $4^3$ の時最大となる山形になっている．これは，PE数がこれより小さい時は送り先のPE数(使用するPE数)によって，これより大きい時は割り付けられたデータによって実行できるバタフライ演算の段数に制限を受けるため，式(2)によって，通信回数が影響を受ける．しがたって，これらの値が同じになる $4^3$ を最大とする山形になるのである．これは，通信時間がスタートアップ時間に左右される事を示している．また，図３より，転送するデータの総量はMulti-stage法を用いた方が小さい事が分かる．

したがって，通信するデータ量を小さくすることが，単純に通信時間が小さくなることに結びつくとは限らず，スタートアップ時間によって，どの方法がよい性能を示すかが決まる．

## ４．　Cray T3Eへの実装

### ４．１．　Cray T3Eの構成

Cray T3EのＰＥノードの構成を，図５に示す．各ＰＥノードは，１つのＰＥとＰＥ間通信のインターフェース（６方向）および，ローカルメモリから構成される．

CPUには，DEC chip21164 Alphaを使用され，動作周期300MHzで動作し，論理演算性能は600MFLOPS/1200MIPSである．

ＰＥ間の通信ネットワークは，図６に示すような３次元トーラス型を採用している．

OSには，UNICOS/mkを採用し，FortranおよびC/C++をサポートしている．通信ライブラリとして，PVMとMPIが使用できる．

### ４．２．処理時間の実測値

ここでは，本アルゴリズムをCray T3Eに実装し，処理時間を実測した結果について述べる．使用したＰＥ数は，６４個である．データ数N=4096($4^6$)の場合のＦＦＴの実行時間を図７に示す．図より，Stage-by-stage法を用い，atomic型の転送を行った場合が，最も高速である．また，図示のようにatomic型の転送の場合には，Multi-stage法，Stage-by-stage法のいずれの場合も，１ＰＥでの実行の場合よりも遅くなってしまっている．buffer型の転送の場合には，P=16までは両者はほぼ同じように減少しているが，P=64の時，Multi-stage法は実行時間が増加してしまっている．

次に，処理時間の内訳を図８に示す．図８(a)と(c)，図８(b)と(d)を比べて，atomic型とbuffer型で通信時間が大きく違うことから，スタートアップ時間が非常に大きいことが示唆される．図８(a),(b)より，atomic型の転送の場合には，バタフライ演算の時間は減少しているものの，通信時間の増加が激しく，全体としては，遅くなってしまっていることがわかる．図８(c)と(d)を比較すると，P=16,P=64の時，Stage-by-stage法ではわずかに減少しているのに対し，Multi-stage法では通信時間が増加していることが分かる．この違いは，先の図３の通信回数の違いより説明できる．Stage-by-stage法では，PE数の増加とともに緩やかに通信回数が増すのに対し，Multi-stage法では，P=64で最大となるように急激に増加している．このため，スタートアップ時間が大きい場合，送信する総データ量が減少しているにもかかわらず，通信時間が増加してしまっているのである．

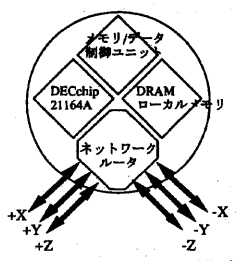


図５ ＰＥノードの構成

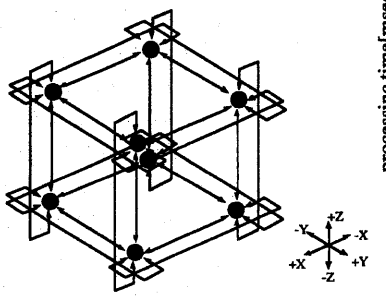


図６ ３次元トーラス型ネットワーク

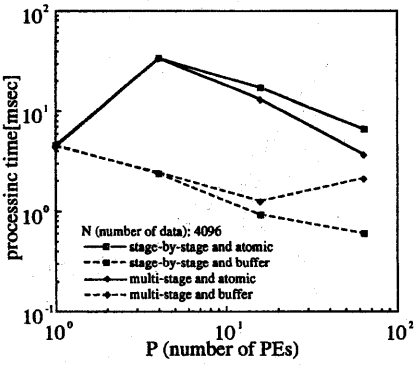
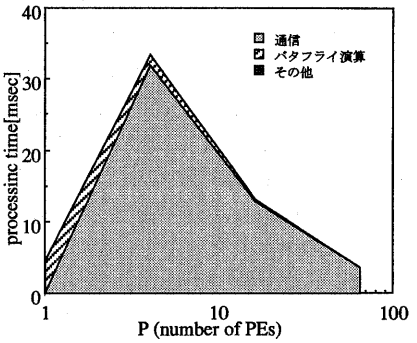
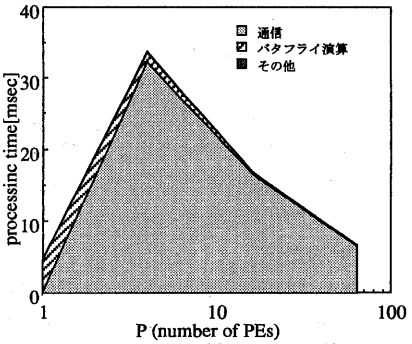


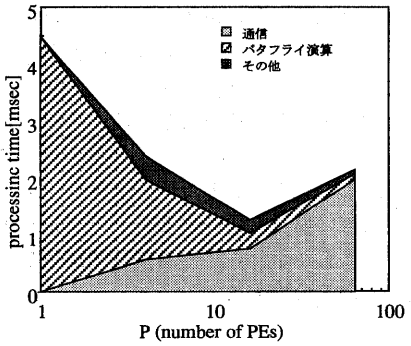
図７ ＦＦＴの実行時間



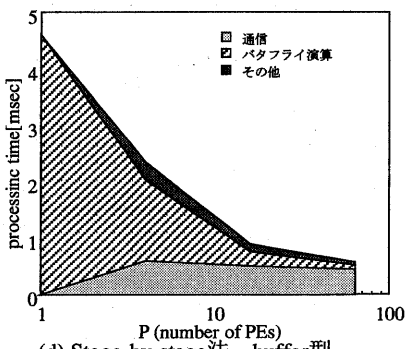
(a) Multi-stage法，atomic型



(b) Stage-by-stage法，atomic型



(c) Multi-stage法，buffer型



(d) Stage-by-stage法，buffer型

図８　ＦＦＴの実行時間の内訳

## ５．むすび

大域通信ネットワークを用いた基数Rの並列ＦＦＴアルゴリズムを説明し，その実装についても述べた．本アルゴリズムを実装する際には，スタートアップ時間の大きさと，転送モードが重要なポイントとなる．もし，atomic型しかサポートしていなければ，Multi-satage方が，buffer型をサポートしていれば，Stage-by-stage法が有利であることを示した．また，Cray T3Eへ実装したときの実測値も示した．

## 参考文献

[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] 仁木, 高橋,"二進木結合型マルチプロセッサによる高速フーリエ変換の並列計算", 電情通学論誌, Vol.J72-D-I, No.4, pp.229-237(1989).

[5] C.R.Jesshope,"The Implementation of Fast Radix 2 Transforms on Array Processors", IEEE Trans. Comput., Vol.C-29, No.1, pp.20-27(1980).

[6] L.H.Jamieson, P.T.Mueller Jr. and H.J.Siegel, "FFT Algorithmfor SIMD Parallel Processing Systems", J. of Parallel and Distributed Computing, Vol.3, pp.48-71(1986).

[7] K.Tanno and T.Taketa and S.Horiguchi, "Parallel FFT algorithms using radix 4 butterfly computation on an eight-neighbor processor array", Parallel Comput. 21, pp.121-136 (1995).

[8] "MasPar Parallel Application Language (MPL) Reference Manual," MasPar Computer Corporation, Document No. 9302-0000(1991).

[9] 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).

[10] "nCUBE Supercomputers," NCUBE Corporation, No. G2-0100(1989).

[11] MPI: A Message-Passing Interface Standard, Message Passing Interface Forum(1995).