並列FFTアルゴリズムの評価*

大橋 直人　武田 利浩　丹野 州宣†
山形大学工学部電子情報工学科 ‡
e-mail {oohashi,takeda,tanno}@etn.yz.yamagata-u.ac.jp

一般に、並列アルゴリズムの研究は、特定のアーキテクチャ上で、ある数のプロセッサを用いて対象とする問題を高速に、効率よく解くことを目的としている。従って、アルゴリズムの評価尺度として速度向上比や効率が用いられてきたが、スケーラビリティーは、プロセッサ効率・プログラムの容易さ・柔軟さ・コスト的効果などについての重要な評価基準である。本論文では、最近提案されている並列アルゴリズムとアーキテクチャを含めて、包括的に性能を評価するISOEFFICIENCYを用いて、8—隣接プロセッサ・アレイ上に構成された並列FFTアルゴリズムのスケーラビリティーを評価する。

1　はじめに

一般に、並列アルゴリズムの研究は、特定のアーキテクチャ上で、ある数のプロセッサを用いて対象とする問題を高速に、効率よく解くことを目的としている。従って、アルゴリズムの評価尺度として速度向上比や効率が用いられており、スケーラビリティーについてはあまり考慮されていなかった。しかしスケーラビリティーは、プロセッサ効率・プログラムの容易さ・柔軟さ・コスト的効果などについての重要な評価基準であり、その測定方法についての報告が多数ある[1]。本論文では、並列アルゴリズムとアーキテクチャを含めて、包括的に性能を評価するものとして、最近提案されているIsoefficiency[2]を用いて、8—隣接プロセッサ・アレイ上に構成された基数2と4の重畳割付法を用いた並列FFTアルゴリズムのスケーラビリティーを評価する。

2　基数2と基数4のFFT

はじめに基数4のFFTついて説明する。離散フーリエ変換G(n)は、N個の複素入力データ列をg(n){0≤n≤N-1}と置くとき、W=exp(-2πj/N)として、次式で表わされる。

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

ここでデータ数を$N=4^m$として、nを4進数

*Evaluation of Parallel FFT(Fast Fourier transform) Algorithm
† Naoto OOHASHI,Toshihiro TAKETA and Kuninobu TANNO
‡ Department of Electrical and Information Engineering,Yamagata University

m桁で表わすとき式(1)は式(2)のようになる。

$$G(n_{m-1},\cdots,n_1,n_0)$$

$$=\sum_{k_0=0}^{3}\sum_{k_1=0}^{3}\cdots\left\{\sum_{k_{m-1}=0}^{3}g(k_{m-1},\cdots,k_1,k_0)W^{4^{m-1}nk_{m-1}}\right\}$$

$$\times W^{4^{m-2}nk_{m-2}}\cdots W^{4nk_1}W^{nk_0} \quad (2)$$

式(2)の{}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。

$$g_1(n_0,k_{m-2},\cdots,k_1,k_0)$$

$$=\sum_{k_{m-1}=0}^{3}g(k_{m-1},\cdots,k_1,k_0)W_1^{n_0k_{m-1}} \quad (3)$$

従って第i段目(2≤i≤m)のFFTの出力として展開すると式(4)のようになる。ただし、$g_i()$はFFTの第i段目の出力を表わし、$h=(4^{m-1}n_0+4^{m-i+1}n_1+\cdots+4^{m-2}n_{i-2})$とする。

$$g_i(n_0,n_1,\cdots,n_{i-1},k_{m-i-1},\cdots,k_1,k_0)$$

$$=\sum_{k_{m-i}=0}^{3}\left\{g_{i-1}(n_0,\cdots,n_{i-2},k_{m-i},\cdots,k_1,k_0)W^{hk_{m-i}}\right\}$$

$$\times W_1^{n_{i-1}k_{m-i}} \quad (4)$$

ここで式(3)、(4)は次のような基数4のバタフライ演算と呼ばれる計算により求められる。ただし$V_l=W^{hl}$、l=0,1,2,3とする。

$$\begin{aligned}
g_i(0)&=g_{i-1}(0)v_0+g_{i-1}(1)v_1+g_{i-1}(2)v_2+g_{i-1}(3)v_3\\
g_i(1)&=g_{i-1}(0)v_0-jg_{i-1}(1)v_1-g_{i-1}(2)v_2+jg_{i-1}(3)v_3\\
g_i(2)&=g_{i-1}(0)v_0-g_{i-1}(1)v_1+g_{i-1}(2)v_2-g_{i-1}(3)v_3\\
g_i(3)&=g_{i-1}(0)v_0+jg_{i-1}(1)v_1-g_{i-1}(2)v_2-jg_{i-1}(3)v_3
\end{aligned} \quad (5)$$

ところでV_0は常に1なので、式(5)は3回の乗算と8回の加算により求められる。

次に基数2のFFTについて説明する。今、データ数を$N=2^m$とするとき、nを2進数m桁で表すと、式(1)は式(6)のようになる。

$$G(n_{m-1},\cdots,n_1,n_0)$$

$$=\sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\cdots\left\{\sum_{k_{m-1}=0}^{1}g(k_{m-1},\cdots,k_1,k_0)W^{2^{m-1}nk_{m-1}}\right\}$$

$$\times W^{2^{m-2}nk_{m-2}}\cdots W^{2nk_1}W^{nk_0} \quad (6)$$

このとき式(6)の{}内を第1段目として展開すると次式のような2点のDFTの式が得られる。

$$g_1(n_0,k_{m-2},\cdots,k_1,k_0)$$

$$=\sum_{k_{m-1}=0}^{1}g(k_{m-1},\cdots,k_1,k_0)W_1^{n_0k_{m-1}} \quad (7)$$

従って第i段目(2≤i≤m)のFFTの出力は、$h=(2^{m-1}n_0+2^{m-i+1}n_1+\cdots+2^{m-2}n_{i-2})$とすると式(8)のようになる。

$$g_i(n_0,n_1,\cdots,n_{i-1},k_{m-i-1},\cdots,k_1,k_0)$$

$$=\sum_{k_{m-1}=0}^{1}\left\{g_{i-1}(n_0,\cdots,n_{i-2},k_{m-i},\cdots,k_1,k_0)W^{hk_{m-i}}\right\}$$

$$\times W_1^{n_{i-1}k_{m-i}} \quad (8)$$

ここで式(7)、(8)は次のような基数2のバタフライ演算により求められる。

$$\begin{aligned}
g_i(0)&=g_{i-1}(0)v_0+g_{i-1}(1)v_1\\
g_i(1)&=g_{i-1}(0)v_0-jg_{i-1}(1)v_1
\end{aligned} \quad (9)$$

同様にV_0は常に1なので、式(9)は2回の乗算と2回の加算により求められる。

3　8—隣接プロセッサ・アレイ

本論文ではFig.1とFig.2に示すような、$P=P_r\times P_r$個の8—隣接と4—隣接のプロセッサ・アレイを仮定する。8—隣接ではFig.1に示す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。一方、4—隣接ではFig.2に示すように各プロセッサは2、3、あるいは4本のリンクを持つ。各リンクの通信は、双方向同時通信と双方向交互通信の場合について、また各PEの転送要求を持つリンクがす

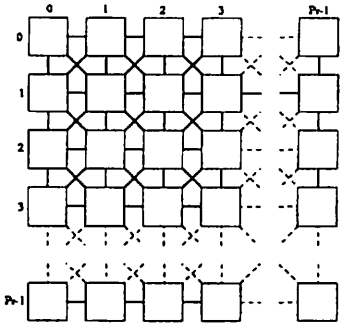

Fig. 1. 8—隣接プロセッサ・アレイの構成

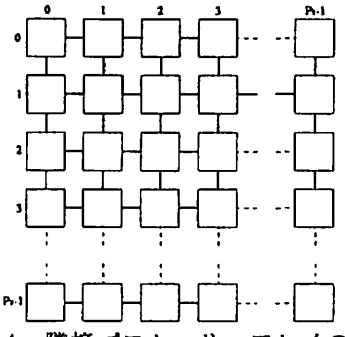

Fig. 2. 4—隣接プロセッサ・アレイの構成

べて同時に並列に転送する全ポート通信と各リンクが順々に連続して転送する1ポート通信も考慮する。各プロセッサへの添え字付けは、8—隣接では、x、y=0,1,…P_r-1とし、それぞれを行、列の番号とおくと、各プロセッサをPE(x,y)あるいはシャフル行主体添え字付けの一連番号z=0,1,…,P-1を用いてPE(z)と表す。4—隣接では、行方向主体の添え字付けの一連番号z'=0,1,…,P-1を用いてPE(z')と表す。

4 並列FFTアルゴリズム

ここでは4—隣接および、8—隣接プロセッサ・アレイ上に構成された基数2と基数4の重畳割付法を用いた並列FFTアルゴリズム[3]について簡単に紹介し、それぞれの並列処理時間を求める。

4.1 重畳割付法を用いた基数4の並列FFTアルゴリズム

まずはじめに、プロセッサに対し、初期データ割り付けを行う重畳割付法について説明する。データ数を$N=N_r\times N_r=4^m$、プロセッサ数を$P=P_r\times P_r=4^q$とする。このとき$P=P_r\times P_r=4^q$であるプロセッサを(m-q)回再帰的に4分割し（Fig.3参照）、下位q桁$n_{m-1}, n_{m-2}, \cdots, n_1, n_0$（ただしnは4進数）が等しいデータを同じPEに重ねて格納する方法が重畳割付法である。つまり、データ$g(n_{q-1}, n_{q-2}, \cdots, n_1, n_0)$を$PE(n_{q-1}, n_{q-2}, \cdots, n_1, n_0)$に割り付ける方法である。データを重畳割り付けした後、Fig.4に示す非インプレイス型演算をq回行う。非インプレイス型演算とは、式(5)で与えられる4点のDFTに必要な4つのデータ$g_{i-1}(0)$、$g_{i-1}(1)$、$g_{i-1}(2)$、$g_{i-1}(3)$を格納しているPE(x,y)が、そのデータを用いてバタフライ演算を行い、その結果$g_i(0)$、$g_i(1)$、$g_i(2)$、$g_i(3)$をFig.4の(c)から(f)に示すように距離r離れた3つのPEに転送する（つまり各PEは$PE(n_0, \cdots, n_{i-2}, 0, k_{q-i-1}, \cdots, k_0)$、$PE(n_0, \cdots, n_{i-2}, 1, k_{q-i-1}, \cdots, k_0)$、$PE(n_0, \cdots, n_{i-2}, 2, k_{q-i-1}, \cdots, k_0)$、$PE(n_0, \cdots, n_{i-2}, 3, k_{q-i-1}, \cdots, k_0)$に対して転送を行う）。これを基数4のバタフライ演算に対する非インプレイス型演算という。q回の非インプレイス型演算後に各プ

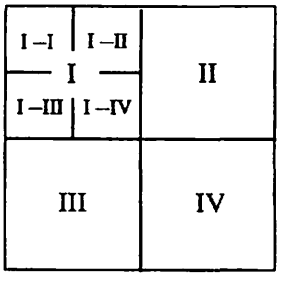

Fig. 3. シャフル行主体順序に番号付けされたプロセッサ・アレイの再帰的4分割

ロセッサは、残りの(q+1)段からm段のバタフライ演算をプロセッサ間通信なしに行う。アルゴリズム1に、この処理を示す。

アルゴリズム1

1) データを重畳割り付けする。

2) i=1～qまで繰り返す。

(a)各PEはデータ$g_{i-1}(n_0, n_1, \cdots, n_{i-2}, k_{m-i}, \cdots, k_1, k_0)$に対し、第i段目のバタフライ演算を行うものである。

(b)その結果$g_i(n_0, n_1, \cdots, n_{i-1}, k_{m-i-1}, \cdots, k_1, k_0)$を$PE(n_0, \cdots, n_{i-1}(=0,1,2,3), k_{q-i-1}, \cdots, k_0)$に転送する。

3) i=(q+1)～mまで繰り返す。

各$PE(n_0, n_1 \cdots, n_{q-1})$はデータ$g_i(n_0, n_1, \cdots, n_{i-1}, k_{m-i-1}, \cdots, k_1, k_0)$を用いてi段目のバタフライを行う。

4.1.1 基数4のFFTアルゴリズムの並列処理時間

P個のプロセッサを用いたときの並列処理時間T_pを求める。基数4のFFTアルゴリズムの並列処理時間T_pは、

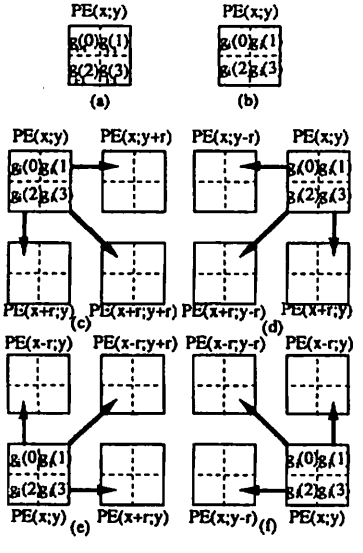

Fig.4. 8—隣接プロセッサ・アレイ上での基数4非インプレイス型演算の様子

t_b：基数4のバタフライ演算時間

t_u：通信のスタートアップ時間

t_c：隣接するプロセッサ間の通信時間

N：データ数（$N=N_r \times N_r$）

P：プロセッサ数（$P=P_r \times P_r$）

D：同時通信の時D=1、
交互通信の時D=4

S：全ポート通信の時S=1、
1ポート通信の時S=3

とおくとき式(10)で表わされる。

$$T_p=\frac{SDN}{4P}\{(P_r-1)t_c+(\log_2 P_r)t_u\}$$
$$+\frac{N}{4P}(\log_2 N_r)t_b \qquad (10)$$

4.2 重畳割付法を用いた基数2の並列FFTアルゴリズム

基数2の並列FFTアルゴリズムは、斜め方向のデータ転送を必要としないので4—隣接プロセッサ・アレイ上に構成される。今、プロセッサの個数を$P=P_r \times P_r=2^q$、データの個数を$N=N_r \times N_r=2^m$とする。このとき、基数2の重畳割付法は、プロセッサを行主体順序で添え字付けをし、下位q桁、つまりデータ$g(n_{q-1}, n_{q-2}, \cdots, n_1, n_0)$を$PE(n_{q-1}, n_{q-2}, \cdots, n_1, n_0)$に割り付ける方法である。データを重畳割り付けした後、Fig.5に示す非インプレイス型演算をq回行う。基数2の非インプレイス型演算とは、式(9)で与えられる2点のDFTに必要な2つのデータ$g_{i-1}(0)$、$g_{i-1}(1)$を格納しているPE(x,y)が、そのデータを用いてバタフライ演算を行い、その結果$g_i(0)$、$g_i(1)$をFig.5に示すように距離r離れたPEに転送する（つまり各PEは次の2つの転送、$PE(n_0, \cdots, n_{i-2}, 0, k_{q-i-1}, \cdots, k_0)$、$PE(n_0, \cdots, n_{i-2}, 1, k_{q-i-1}, \cdots, k_0)$を行う）。はじめのq/2回の非インプレイス型演算では、Fig.5の(a)、(b)のように列方向に転送を行い、次のq/2回では(c)、(d)のように行方向に転送を行

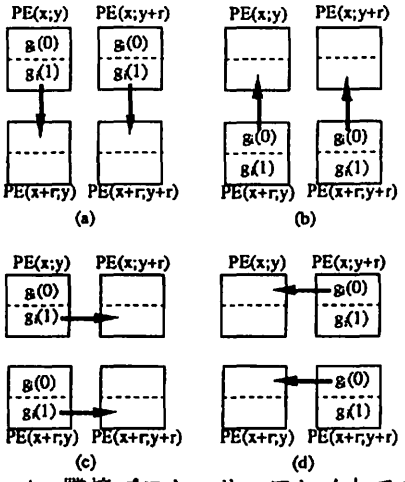

Fig. 5. 4一隣接プロセッサ・アレイ上での基数2非インプレイス型演算の様子

う。そうして、q段のバタフライ演算後に各プロセッサは、残りの(q+1)段からm段までのバタフライ演算をプロセッサ間通信なしに行う。アルゴリズム2に、この処理を示す。

アルゴリズム2

1) データを重畳割り付けする。

2) i=1～q/2まで繰り返す。

(a)各PEはデータ$g_{i-1}(n_0, n_1, \cdots, n_{i-2}, k_{m-i}, \cdots, k_1, k_0)$に対し、第i段目のバタフライ演算を行う。

(b)その結果$g_i(n_0, n_1, \cdots, n_{i-1}, k_{m-i-1}, \cdots, k_1, k_0)$を$PE(n_0, \cdots, n_{i-1}(=0,1), k_{q-i-1}, \cdots, k_0)$に転送する。

2) i=(q/2+1)～qまで繰り返す。

(a)各PEはデータ$g_{i-1}(n_0, n_1, \cdots, n_{i-2}, k_{m-i}, \cdots, k_1, k_0)$に対し、第i段目のバタフライ演算を行う。

(b)その結果$g_i(n_0, n_1, \cdots, n_{i-1}, k_{m-i-1}, \cdots, k_1, k_0)$を$PE(n_0, \cdots, n_{i-1}(=0,1), k_{q-i-1}, \cdots, k_0)$に転送する。

3) i=(q+1)～mまで繰り返す。

各$PE(n_0, n_1 \cdots, n_{q-1})$はデータ$g_i(n_0, n_1, \cdots, n_{i-1}, k_{m-i-1}, \cdots, k_1, k_0)$を用いてi段目のバタフライを行う。

4.2.1 基数2のFFTアルゴリズムの並列処理時間

P個のプロセッサを用いた並列処理時間T_p'を求める。基数2のFFTアルゴリズムの並列処理時間T_p'は、

t_b'：基数2のバタフライ演算時間

D'：同時通信の時D'=1、
　　交互通信の時D'=2

とするとき式(11)で表わされる（ただしt_b'、D'以外のパラメータは4.1.1と同じものとする）。

$$T_p' = \frac{D'N}{P}\{(P_r-1)t_c + (\log_2 P_r)t_u\} + \frac{N}{P}(\log_2 N_r)t_b' \qquad (11)$$

5 ISOEFFICIENCY

一般に、並列アルゴリズムを実装したマルチプロセッサ・システムでは、プロセッサの個数に比例した速度向上を得るのは難しいことである。つまり1個のプロセッサを用いた時の速度向上は1になるが、それより多くのプロセッサを用いても速度向上比は常にプロセッサ数よりも少ないものになってしまう。Fig.6はn整数の加算を行う並列アルゴリズム

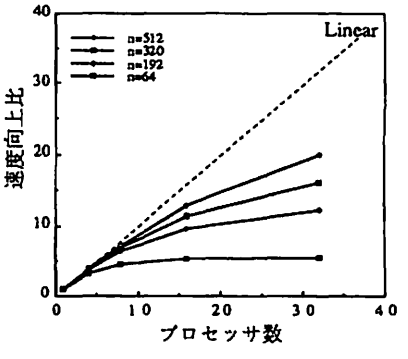

Fig. 6. n数の加算の速度向上比

	p=1	p=4	p=8	p=16	p=32
n=64	1.0	.80	.57	.33	.17
n=192	1.0	.92	.80	.60	.38
n=320	1.0	.95	.87	.71	.50
n=512	1.0	.97	.91	.80	.62

Fig.7. n数の加算の効率

の速度向上とプロセッサ数との関係の一例を示したものであるが、速度向上比はプロセッサ数に対して線形に増加せずに、飽和する傾向にあることが分かる。プロセッサの個数が等しい場合には、問題のサイズが大きくなるにつれ、優れた速度向上を示すことが分かる。従って、Fig.7からも言えるように、プロセッサ数が増えると効率が落ち、問題サイズが大きくなると効率が上がることから、次のような疑問が生じる。

「効率を固定したまま、プロセッサ数の増加と同じ比率で速度向上を増加させるには、どのくらいの割合で問題サイズを大きくするべきか？」

以上のことを背景として提案されたのが、Isoefficiencyである。Isoefficiencyは、並列アルゴリズムを実装したマルチプロセッサ・システムにおいて、プロセッサの数が増加する時に、ある一定の効率と速度向上比を維持するために必要な問題サイズを示す関数である。

5.1 ISOEFFICIENCYの定義

T_1：単一プロセッサ上でのアルゴリズムの逐次処理時間

T_p：P個のプロセッサ上での並列アルゴリズムの処理時間

T_o：並列アルゴリズムにおける通信などのオーバーヘッド

S：速度向上

P：プロセッサ数

以上のように仮定すると効率Eは、

$$E=\frac{S}{P}=\frac{1}{1+T_o/T_1} \tag{12}$$

となる。

ここで問題サイズをW、T_1における単位演算時間をt_bとすると、$T_1=Wt_b$となるので、これを用いて効率EをWについて、次のように書き直すことが出来る。

$$W=\frac{1}{t_b}\left(\frac{E}{1-E}\right)T_o \tag{13}$$

問題サイズWについての上記の方程式がIsoefficiency関数でる。

5.2 ISOEFFICIENCYによるアルゴリズムの評価

ここでは8一隣接プロセッサ・アレイ上に構成された重畳割付法によるFFTアルゴリズムのIsoefficiencyを求め、その結果からアルゴリズムの評価を行う。

はじめに基数4の並列FFTアルゴリズムのIsoefficiencyを求める。基数4のアルゴリズムの問題サイズWは$\frac{N}{4}\log_4 N$なので、単一プロセッサ上でのアルゴリズムの逐次処理時間T_1は式(14)で示される。

$$T_1=t_b\times\left(\frac{N}{4}\log_2 N\right)=t_b\times\left(\frac{N}{4}\log_2 N_r\right) \tag{14}$$

このとき、Pプロセッサでの並列処理時間は既に式(10)で与えられているので、並列アルゴリズムのオーバーヘッドが、式(15)として求められる。

$$T_o=SDN\{(P_r-1)t_c+(\log_2 P_r)t_u\} \tag{15}$$

従って全体のIsoefficiency関数は式(13)を用いて式(16)のようになる。

$$W=\frac{1}{4}\times\frac{t_c}{t_b}\times\frac{E}{1-E}SDP_r4^{SDP_r\frac{t_c}{t_b}\times\frac{E}{1-E}} \tag{16}$$

次に基数2の並列FFTアルゴリズムのIsoefficiency関数を求める。問題サイズは$\frac{N}{2}\log_2 N$なので、単一プロセッサでのアルゴリズムの逐次処理時間は式(17)のようになる。

$$T_1=\frac{1}{2}t_b' N\log_2 N \qquad (17)$$

P個のプロセッサを用いたときのアルゴリズムの並列処理時間が式(11)で与えられているので、オーバーヘッドは式(18)として求められる。

$$T_o=D'N\{(P_r-1)t_c+(\log_2 P_r)t_u\} \qquad (18)$$

従って全体のIsoefficiency関数は式(19)で表わされる。

$$W=\frac{t_c}{t_b'}\times\frac{E}{1-E}\times D'P_r 2^{2D'P_r\frac{t_c}{t_b'}\times\frac{E}{1-E}} \qquad (19)$$

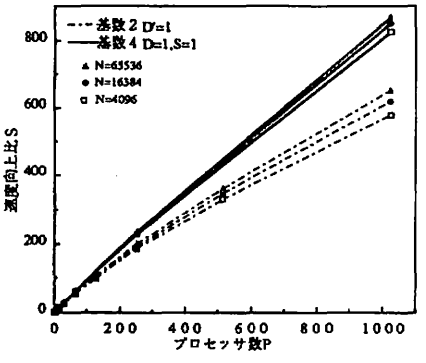

Fig. 8. 各問題サイズにおける速度向上比

以上の結果を用いて、全ポート通信・同時通信を仮定したうえでの基数2と4の並列FFTアルゴリズムの速度向上比をFig. 8に、Isoefficiency特性をFig.9に示す。Fig. 8から速度向上比は、基数2に比べ基数4の方が優れていることがわかる。先に述べたように、問題のサイズが一定のときプロセッサ数が増加するにつれ速度向上が飽和する傾向にある一方で、プロセッサ数が一定のときには問題サイズが大きい方が速度向上が良いことがわかる。プロセッサに対して線形な速度向上を維持するためには、Fig.9に示されるIsoefficiencyに従って問題のサイズを増加させればよい。またFig.9からIsoefficiency特性についても基数4の方が基数2より優れている。すなわち、プロセッサ数を増加させた時、効率を一定に保ち、プロセッサ数に対して速度向上を線形に保つためには、基数2の方が基数4より問題のサイズを急速に増加させる必要がある。従って全ポート通信・同時通信の場合において、基数4のFFTアルゴリズムは基数2のアルゴリズムに比べて処理速度と効率だけでなく、スケーラビリティーについても優れているといえる。

次に1ポート通信・交互通信を仮定した場

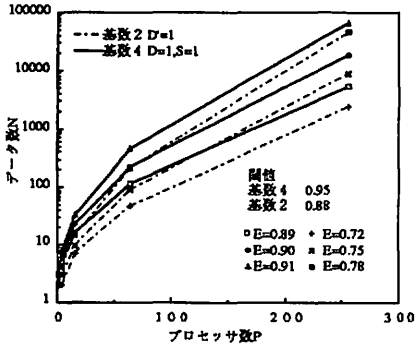

Fig. 9. Isoefficiency特性

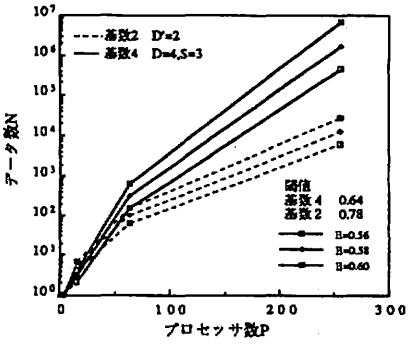

Fig. 10. Isoefficiency特性

合の基数2と4のアルゴリズムのIsoefficiency特性をFig.10に示す。このグラフの結果から、プロセッサ数が少ない場合には基数4のIsoefficiencyが小さいが、プロセッサ数が増加するにつれ、基数4に比べ基数2のアルゴリズムのIsoefficiencyの方が優れたものとなっている。また効率の閾値についても、全ポート・同時通信の場合とは異なり、基数2の方が良好である。従って1ポート通信・交互通信の場合、基数4のアルゴリズムは基数2に比べ25%乗算数が減少しているにも関わらず、アーキテクチャの通信方式がアルゴリズムの長所をいかせないため、スケーラビリティーが極端に悪くなっている。

6 結び

Isoefficiencyに基づいた基数2と4の並列FFTアルゴリズムのスケーラビリティーを評価した。全ポート通信・同時通信の場合は、基数4のアルゴリズムはIsoefficiency特性も良好で、効率の閾値も非常に高い。それに対して、1ポート通信・交互通信の場合になると基数4のIsoefficiency特性は、基数2のアルゴリズムより悪い。このことから、基数4のアルゴリズムがアーキテクチャの通信方式に大きく影響を受けることが分かった。

理想的なFFTにおいてプロセッサ数が$\Theta(P\log P)$で増加するとき、効率を一定に保つためには、問題サイズを$\Theta(P\log P)$で増加させれば良い（つまりFFTにおけるIsoefficiencyの下限は$\Theta(P\log P)$）のに対して、基数4のFFTアルゴリズムは4の冪乗のオーダーで問題のサイズを増加させなければならない。ハイパーキューブ上のFFTで、Isoefficiencyのオーダーが$\Theta(P t_s \frac{E}{1-E} \log P)$という報告[4]もあることから、4の冪乗のオーダーはだいぶ高いものと思われる。

今後はスケーラビリティーも十分に考慮しながら8—隣接プロセッサ・アレイ上での基数4と基数2のアルゴリズムを多次元FFTへ拡張したい。

参考文献

[1]D.Nussbaum and A.Agarwal, "Scalability of parallel machines," Commun.ACM, vol.34, no.3, pp.57-61(1991).

[2]Ananth Y.Grama,Anshul Gupta,and Vipin Kumar, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures, "IEEE Parallel & Distributed Technology, Vol.1, No3, pp.12-21(1993).

[3]K.Tanno, T.Taketa and and S.Horiguchi, "Parallel FFT Algorithms using Radix 4 Butterfly Computation on an 8-Neighbor Processor Array, "to be appeared in Parallel Computing,vol.20(1994).

[4]A.Gupta and V.Kumar, "The Scalability of FFT on Parallel Computers, "IEEE Trans.Parallel and Distributed Systems, Vol.4, No.7(1993).