ハイパフォーマンス
コンピューティング 62−20
（1996. 8. 29）

分散メモリ型並列計算機による 2, 3, 5 基底の FFT の実現と評価

高橋大介† 金田康正††

東京大学大学院理学系研究科情報科学専攻†
東京大学大型計算機センター††

本稿では，分散メモリ型並列計算機において，2, 3, 5 基底の FFT を実現し，その性能を評価した結果について述べる．

FFT アルゴリズムは，行列の分解に帰着する．この行列の分解の考え方を 2, 3, 5 基底の FFT に適用し，さらに並列 FFT アルゴリズムに拡張出来ることを示す．また具体的な 2, 3, 5 基底の FFT アルゴリズムを示す．

この並列 FFT アルゴリズムを MIMD 型分散メモリ型並列計算機 HITACHI SR2201 上に実現し，性能評価を行った．実現にはメモリコピーの発生しない高速なリモート DMA 転送を用いた結果，2^{28}点 FFT において最大 14.0GFLOPS という性能が得られた．

A radix-2, 3, 5 FFT on Distributed Memory Parallel Computers

Daisuke TAKAHASHI† Yasumasa KANADA††

Department of Information Science, Graduate School of Science, University of Tokyo†
Computer Centre, University of Tokyo††

This paper describes how the radix-2, 3, 5 fast Fourier transform (FFT) was implemented and evaluation on the distributed memory parallel computers. The FFT algorithm is derived by means of matrix factorization. Basic algorithms for the radix-2, 3, 5 are shown. For the implementation, memory copy free, fast remote DMA data transfer mechanism was used. According to the experimental results with distributed memory MIMD parallel computer, HITACHI SR2201 of 256 PE, about 14.0GFLOPS at 2^{28} point FFT were attained.

1 はじめに

本稿では，分散メモリ型並列計算機により，2, 3, 5 基底の FFT を実現し，評価した結果について述べる．

FFT アルゴリズムは，行列の分解に帰着出来ることが知られている．この行列分解の考え方を 2, 3, 5 基底の FFT に適用し，さらに並列 FFT アルゴリズムに拡張出来ることを示す．この並列 FFT アルゴリズムを MIMD 型分散メモリ型並列計算機 HITACHI SR2201 上に実現し，性能評価を行う．

以下，2 章で高速フーリエ変換について，3 章で並列 FFT アルゴリズムについて，4 章で本稿で示すアルゴリズムの評価結果を示す．

2 高速フーリエ変換

高速フーリエ変換 (FFT)[5] は，離散フーリエ変換 (DFT) を高速に計算するアルゴリズムとして知られている．

$$F_j(x) = \sum_{k=0}^{n-1} x_k e^{-2\pi ijk/n}, \quad 0 \le j < n \qquad (1)$$

$$F_j^{-1}(x) = \frac{1}{n} \sum_{k=0}^{n-1} x_k e^{2\pi ijk/n}, \quad 0 \le j < n \qquad (2)$$

n が $n = lm$ と分解出来るものとすると，式 (1),(2) における j および k は，$j = pm+q$, $k = rl+s$ と書くことができる．ここで，p, s は $0 \sim l-1$ の値をとり，q, r は $0 \sim m-1$ の値をとるものとする．そうすると，式 (1) は次のように書くことが出来る．

$$\begin{aligned} & F_{pm+q}(x) \\ &= \sum_{s=0}^{l-1}\sum_{r=0}^{m-1} x_{rl+s} e^{-2\pi i(pm+q)(rl+s)/n} \\ &= \sum_{s=0}^{l-1}\sum_{r=0}^{m-1} x_{rl+s} e^{-2\pi ipms/n} e^{-2\pi iqrl/n} e^{-2\pi iqs/n} \\ &= \sum_{s=0}^{l-1}\sum_{r=0}^{m-1} x_{rl+s} e^{-2\pi ips/l} e^{-2\pi iqr/m} e^{-2\pi iqs/n} \\ &= \sum_{s=0}^{l-1} \left[\sum_{r=0}^{m-1} (x_{rl+s} e^{-2\pi iqs/n}) e^{-2\pi iqr/m} \right] e^{-2\pi iqs/l}, \\ & \qquad\qquad (0 \le p < l,\ 0 \le q < m) \end{aligned}$$

上式は，n 点 DFT が，l 点 DFT と m 点 DFT に分解されることを示している．

これから，例えば，$n = n_1 n_2$ と分解されるとき，次に示すような "four step" FFT アルゴリズムとして知られている方法が導かれる [1].

1. n_1 組の n_2 点 FFT を $n_1 \times n_2$ の複素数行列に対して行う．
2. $n_1 \times n_2$ 行列 A_{jk} を考え，それに $e^{-2\pi ijk/n}$ を掛ける．
3. $n_1 \times n_2$ 行列を $n_2 \times n_1$ 行列に転置する．
4. n_2 組の n_1 点 FFT を $n_2 \times n_1$ の複素数行列に対して行う．

実際のインプリメントにおいては，n_2 点 FFT を計算する際，n_1 組の FFT を最内側のループにもっていくとひねり係数 $e^{-2\pi ijk/n}$ が共通になるので，効率が良くなる．同様に，n_1 点 FFT を計算する際も，n_2 組の部分を最内側のループにもっていくようにする．

3 並列 FFT アルゴリズム

並列 FFT アルゴリズムを考えるにあたって，先ほど述べた，"four step" FFT の考え方を適用する．

データの長さを $N = N_1 N_2 N_3$, P をプロセッサ数とした場合，次の手順で FFT を計算することが出来る．

ここで，$W_N = \exp(-2\pi i/N)$ とする．

1. $N_1 \cdot N_2 \cdot N_3$ の入力データに対して，各プロセッサでは，$N_1 \cdot N_2 \cdot (N_3/P)$ のデータを分散して持つ (ブロック分割)．
2. $N_1 \cdot N_2 \cdot (N_3/P)$ のデータをプロセッサ間で交換して $N_3 \cdot N_1 \cdot (N_2/P)$ のデータを持つようにする．
3. $N_1 \cdot (N_2/P)$ 組の N_3 点 FFT を行う．
4. 結果のデータを $X(j_1, j_2, k_3)$ とするとき，$W_N^{(j_2 N_1 + j_1)k}$ を掛ける．
5. $N_3 \cdot N_1 \cdot (N_2/P)$ のデータをプロセッサ間で交換して $N_3 \cdot N_1 \cdot (N_2/P)$ のデータを持つようにする．
6. $N_1 \cdot (N_3/P)$ 組の N_2 点 FFT を行う．
7. 結果のデータを $X(j_1, k_2, k_3)$ とするとき，$W_{N_1 N_2}^{j_1 k_2}$ を掛ける．
8. 各プロセッサ内で $N_1 \times N_2$ 行列を $N_2 \times N_1$ 行列に転置する．
9. $N_2 \cdot (N_3/P)$ 組の N_1 点 FFT を行う．

10. $N_3 \cdot N_1 \cdot (N_2/P)$ のデータをプロセッサ間で交換して $N_1 \cdot N_2 \cdot (N_3/P)$ のデータを持つようにする．

2. の部分で，$N_1 \cdot N_2 \cdot (N_3/P)$ のデータをプロセッサ間で交換しているのは，計算すべきデータを各プロセッサ内に持ってくる必要があるためである．5., 10. の部分についても同様である．この通信は，全対全通信のパターンとなる．

今回は，ブロック分割の場合について評価を行ったが，サイクリック分割にした場合には，最初から計算すべきデータが各プロセッサ内にあるため，2. の部分のデータ交換が不要になる．

したがって，ブロック分割では3回必要なプロセッサ間のデータ交換が，サイクリック分割では2回で済むので，通信時間を2/3にすることが出来る．

上記のアルゴリズムにおいては，N_2 および N_3 がプロセッサ数 P で割り切れれば，2のべきで無くても良いことが分かる．また，$N = N_1N_2N_3$ であるので，N が P^2 で割り切れるように N および P を選択すれば良いことになる．

3.1 プロセッサ内の逐次 FFT アルゴリズム

基数 2, 3, 5 の FFT アルゴリズムとしては，Singleton[2] によるものや，Temperton[3] による Mixed-Radix FFT が知られている．

今回は，フーリエ変換すべきデータ数 $N = 2^p3^q5^r$ において，"four step" FFT アルゴリズムを適用した．

まず，2^p3^q 点 FFT を "four step" FFT アルゴリズムにより計算し，その結果と 5^r 点 FFT の結果を "four step" FFT アルゴリズムで統合する．実際のインプリメントにおいては，2^p 点 FFT を計算する際，3^q5^r 組の FFT を最内側のループにもっていくとひねり係数が共通になり，効率が良くなる．同様に，3^q 点 FFT を計算する際も，2^p5^r 組の部分を最内側のループにもっていくようにする．

基数2の FFT アルゴリズムとしては，Stokham のアルゴリズム [4] を用いた．

Stokham のアルゴリズムは，Cooley-Tukey のアルゴリズム [5] に比べて，入力と出力が重ね書きできないために，メモリは2倍必要となる．しかし，最内側のループの配列アクセスが連続的になるので，キャッシュのヒット率が良くなるという特徴がある．また，ベクトルプロセッサにも適しているアルゴリズムとしても知られている．

基数3の FFT アルゴリズムは，Stokham のアルゴリズムを拡張して，Rader の Small-n アルゴリズム [6] を，基数3の場合について適用した．

基数5の FFT アルゴリズムは，基数3と同様に，Stokham のアルゴリズムを拡張して，Rader の Small-n アルゴリズムを，基数5の場合について適用した．Small-n アルゴリズムとしては，他にも Winograd による WFTA(Winograd Fourier Transform Algorithm)[7] がある．WFTA の Small-n による基数5の変換は，Rader のアルゴリズムに比べると，乗算は2回減るが，加算が2回増えてしまう．並列計算機の各プロセッサにおいて，乗算のサイクル数が加算に比べて多い場合には，WFTA の Small-n アルゴリズムが有効であるが，今回評価に使用したプロセッサ (SR2201) は，積和計算を1サイクルで実行できるアーキテクチャであるため，所要サイクル数は，加算の回数で決まる．したがって，このような場合については，Rader のアルゴリズムが高速に実行できることが分かる．

また，基数2の FFT においては，演算回数の少ない基数4, 8の FFT を部分的に適用することにより，効率を高くすることができる [8]．同様にして，基数 $6(=2\times3)$ や基数 $9(=3\times3)$ なども適用可能であり，演算回数をより少なくすることが出来るが，基数を大きくするにしたがって，アルゴリズムが複雑となるので，基数 2, 3, 4, 5, 6 の組み合わせが，実用上限度とされている [3]．

今回は，基数 2, 3, 5 の組み合わせで，実現および評価を行った．

3.2 基数2の FFT

基数2における，Stokham のアルゴリズムについて説明する．

$n = 2lm$ とする．ここで l および m は2のべきであるものとする．l の初期値は $n/2$ とし，反復ごとに2で割っていく．m の初期値は1とし，反復ごとに2倍していく．配列 X は，入力データであり，Y は出力データである．実際にインプリメントするときには，反復において，X から Y，Y から X に出力されるようにすると，メモリコピーを減らすことができる．ここで，$\omega_p = e^{-2\pi i/p}$ である．

$$
\begin{aligned}
c_0 &= X(k+jm) \\
c_1 &= x(k+jm+lm) \\
Y(k+2jm) &= c_0 + c_1
\end{aligned}
$$

$$
\begin{aligned}
Y(k+2jm+m) &= \omega_{2l}^{j}(c_0-c_1) \\
0\le j<l \qquad & \qquad 0\le k<m
\end{aligned}
$$

3.3 基数3のFFT

基数3の場合に拡張した，Stokhamのアルゴリズムは次のようになる．

$n=3lm$ とする．ここで l および m は3のべきであるものとする．l の初期値は $n/3$ とし，反復ごとに3で割っていく．m の初期値は1とし，反復ごとに3倍していく．ここで，$\omega_p=e^{-2\pi i/p}$ である．

$$
\begin{aligned}
c_0 &= X(k+jm) \\
c_1 &= X(k+jm+lm) \\
c_2 &= X(k+jm+2lm) \\
d_0 &= c_1+c_2 \\
d_1 &= c_0-\frac{1}{2}d_0 \\
d_2 &= -i\left(\sin\frac{\pi}{3}\right)(c_1-c_2) \\
Y(k+3jm) &= c_0+d_0 \\
Y(k+3jm+m) &= \omega_{3l}^{j}(d_1+d_2) \\
Y(k+3jm+2m) &= \omega_{3l}^{2j}(d_1-d_2) \\
0\le j<l \qquad & \qquad 0\le k<m
\end{aligned}
$$

3.4 基数5のFFT

基数5の場合に拡張した，Stokhamのアルゴリズムは次のようになる．

$n=5lm$ とする．ここで l および m は5のべきであるものとする．l の初期値は $n/5$ とし，反復ごとに5で割っていく．m の初期値は1とし，反復ごとに5倍していく．ここで，$\omega_p=e^{-2\pi i/p}$ である．

$$
\begin{aligned}
c_0 &= X(k+jm) \\
c_1 &= X(k+jm+lm) \\
c_2 &= X(k+jm+2lm) \\
c_3 &= X(k+jm+3lm) \\
c_4 &= X(k+jm+4lm) \\
d_0 &= c_1+c_4 \\
d_1 &= c_2+c_3 \\
d_2 &= \left(\sin\frac{2\pi}{5}\right)(c_1-c_4) \\
d_3 &= \left(\sin\frac{2\pi}{5}\right)(c_2-c_3) \\
d_4 &= d_0+d_1 \\
d_5 &= \frac{\sqrt{5}}{4}(d_0-d_1) \\
d_6 &= c_0-\frac{1}{4}d_4 \\
d_7 &= d_6+d_5 \\
d_8 &= d_6-d_5 \\
d_9 &= -i\left(d_2+\frac{\sin(\pi/5)}{\sin(2\pi/5)}d_3\right) \\
d_{10} &= -i\left(\frac{\sin(\pi/5)}{\sin(2\pi/5)}d_2-d_3\right) \\
Y(k+5jm) &= c_0+d_4 \\
Y(k+5jm+m) &= \omega_{5l}^{j}(d_7+d_9) \\
Y(k+5jm+2m) &= \omega_{5l}^{2j}(d_8+d_{10}) \\
Y(k+5jm+3m) &= \omega_{5l}^{3j}(d_8-d_{10}) \\
Y(k+5jm+4m) &= \omega_{5l}^{4j}(d_7-d_9) \\
0\le j<l \qquad & \qquad 0\le k<m
\end{aligned}
$$

3.5 演算回数

基数2, 3, 5のFFTにおける実数演算回数の比は，表1のようになる．

表1. 基数2, 3, 5のFFTにおける実数演算回数の比

n	加算	乗算
2	1.5	1
3	2.667	2
5	4	2.8

これから，$N=2^p3^q5^r$ とした場合のFFTにおける加算の回数 $A(N)$ と乗算の回数 $M(N)$ は，

$$
\begin{aligned}
A(N) &= 2N(1.5p+2.667q+4r-1)+2 \\
M(N) &= 2N(p+2q+2.8r-2)+4
\end{aligned}
$$

となる．

3.6 演算時間

$N=2^p3^q5^r$ 点FFTの計算量は，$A(N)+M(N)=2N(2.5p+4.667q+6.8r-3)+6$ となる．

プロセッサ単体の平均演算性能を S_{1dim}(MFLOPS)，プロセッサ数を P とすると，演算時間 T_{1dim} は，

$$
T_{1dim}=\frac{2N(2.5p+4.667q+6.8r-3)+6}{P\cdot(S_{1dim}\times 10^6)}\ (\mathrm{sec})
$$

となる．

3.7 通信時間

プロセッサ間のデータの交換において，全PE対全PEの通信が発生する．

この通信においては，各プロセッサは他の全てのプロセッサに対し，N/P^2個の複素数倍精度配列，つまり $16\cdot(N/P^2)$ バイトのデータを送信する．

したがって，通信時間 T_{comm}は，Mを通信回数，Pをプロセッサ数，L をレイテンシー (μ s)，Nをデータ数，Bをデータ転送速度 (MB/s) とすると，

$$T_{comm} = M(P-1) \times \left(L + \frac{16\cdot(N/P^2)}{B} \right) \text{ (sec)}$$

と表される．

ブロック分割をした場合，プロセッサ間でデータを 3 回交換することから，$M=3$ であり，サイクリック分割をした場合は，$M=2$ となる．

3.8 コピー時間

各プロセッサ内における行列の転置において，$2\times16\times(N/P)$ バイトのデータをコピーする．コピー性能を S_{copy}(MB/s) とすると，コピー時間 T_{copy}は，

$$T_{copy} = \frac{2\times16\times N}{P\cdot(S_{copy}\times10^6)} \text{ (sec)}$$

となる．

3.9 合計時間

全実行時間 T_{total}は，

$$T_{total} = T_{1dim} + T_{comm} + T_{copy} \text{ (sec)}$$

となる．

4 並列 FFT の性能評価

並列化の評価に際しては，$N=2^p3^q5^r$ の p, q, r およびプロセッサ数 Pを変化させて複素順 FFT と複素逆 FFT の合計の実行時間を測定することにより行った．なお，FFT の計算は倍精度複素数で行っている．

並列計算機としては，MIMD 型並列計算機 HITACHI SR2201(256PE, 総主記憶 64GB) を用いた．計算時間の測定に際しては，256PE すべてをシングルユーザーで使用し，経過時間を測定した．

通信ライブラリとしては，リモート DMA 転送 [9] を用いた．プログラムは，全て FORTRAN で記述し，コンパイラは，日立の最適化 FORTRAN77 V02-01 を用い，最適化オプションとして-W0,'pvec,opt(o(s),fold(2),prefetch(1),ischedule(3),rapidcall(1))' を指定した．

利用可能な主記憶容量の関係でデータ数は，$N=2^{14}\cdot3\cdot5 \sim 2^{26}$ まで変化させ，プロセッサ数を $P=1\sim256$ と変化させて測定した．

結果は表 2 のようになった．表 2 において，* となっているのは，主記憶容量の不足のために実行できなかったことを示している．

表 2. 並列 FFT の実行時間 (単位 sec)

$P \setminus N$	$2^{14}\cdot3\cdot5$	2^{18}	$2^{15}\cdot5^3$	2^{22}	$2^{18}\cdot3^5$	2^{26}
1	1.519	0.877	*	*	*	*
2	0.783	0.451	*	*	*	*
4	0.391	0.218	6.418	3.757	*	*
8	0.216	0.123	3.283	2.029	*	*
16	0.111	0.0631	1.668	1.048	*	*
32	0.0651	0.0372	0.887	0.581	*	*
64	0.0412	0.0283	0.444	0.261	6.567	4.768
128	0.0452	0.0344	0.254	0.159	3.466	2.643
256	0.0649	0.0562	0.170	0.119	1.767	1.183

表 3. データ数の違いによる実行時間の変化 (256PE)

N	time(sec)
$2^{16}\cdot3\cdot5=983040$	0.0852
$2^{20}=1048576$	0.0680
$2^{17}\cdot3^2=1179648$	0.0898
$2^{17}\cdot5^3=16384000$	0.504
$2^{24}=16777216$	0.317
$2^{17}\cdot3^3\cdot5=17694720$	0.512
$2^{20}\cdot3^5=254803968$	6.637
$2^{28}=268435456$	5.378

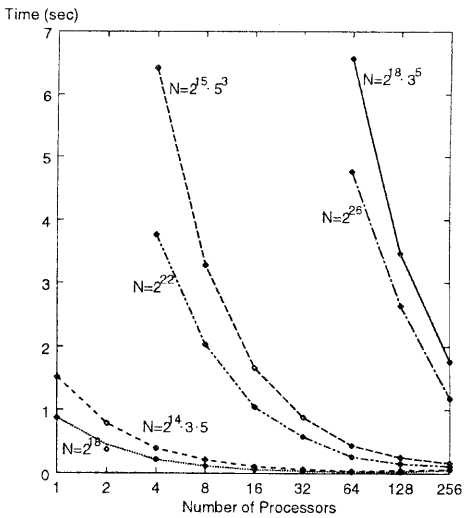

図 1. 並列 FFT の実行時間

図 1 から分かるように，2^{18}点 FFT においては，64 台まではプロセッサ数の増加に伴い実行時間が短縮されているが，128，256 台と増加すると，逆に実行時間が増えてしまっている．

今回の並列 FFT の実現に際しては，メモリコピーの発生しない高速なリモート DMA 転送を用いており，データ量が 1KB 程度で通信スループットが 250MB/s 以上となり，通信の立上り時間も約 5μs と小さくなっているが，．プロセッサ数が増加するに従って，一度に送るデータ量がプロセッサ数の自乗に反比例して小さくなるため，通信の立ち上がり時間が無視できなくなってくるためと考えられる．

表 3 にプロセッサ数を 256 とした場合について，データ数 $N = 2^p 3^q 5^r$ を変化させた場合の順 FFT と逆 FFT を合計した実行時間を示す．

表 4. 理論計算量 $n \log_2 n$ に対する実演算量の比 (相対効率)[2]

基数	相対効率
2	0.500
4	0.375
8	0.333
3	0.631
5	0.689
7	0.763

表 3 では，データ数が 2 のべきだけからなる場合の FFT が，実行時間が短くなっていることが分かる．これは，表 4 からも分かるように，基数 3, 5 の FFT は，基数 2 の FFT に比べて効率が悪くなっているためである．したがって，基数 2, 3, 5 を組み合わせて FFT を計算する際には，2 のべきの部分をなるべく多くして，効率の良い基数 2 の割合を高くすると，性能を向上させることができる．表 3 において，$N = 2^{28}$ 点 FFT(順 FFT +逆 FFT) の実行時間は 5.378 秒となっているが，これは約 14.0GFLOPS に相当する．今回の実現においては，データ数が 2 のべきの場合においては基数 2 のアルゴリズムを用いたが，より高速な基数 4, 8 のアルゴリズムを用いることにより，さらに性能を向上させることができると考えている．

5 まとめ

本稿では，分散メモリ型並列計算機において，2, 3, 5 基底の FFT を実現し，評価した結果について述べた．

256 プロセッサの MIMD 型並列計算機 HITACHI SR2201(理論ピーク性能 76.8GFLOPS) を用いて FFT の計算を行った結果，2^{28} 点 FFT において最大 14.0GFLOPS という性能が得られた．

今後の課題としては，プロセッサ数を増やした場合についての実行時間の詳細な解析や性能向上，そして他の並列計算機への実装および評価が挙げられる．

参考文献

[1] Bailey, D.H.: FFTs in External or Hierarchical Memory, *The J. Supercomputing*, Vol. 4, pp. 23–35 (1990).

[2] Singleton, R.C.: An algorithm for computing the mixed radix Fast Fourier Transform, *IEEE Trans. Audio Electroacoust.*, Vol. 17, pp. 93–103 (1969).

[3] Temperton, C.: Self-sorting mixed-radix fast Fourier transforms, *J. Comput. Phys.*, Vol. 52, No. 1, pp. 1–23 (1983).

[4] Swarztrauber, P.N.: FFT Algorithms for Vector Computers, *Parallel Computing*, Vol. 1, pp. 45–63 (1984).

[5] Cooley, J.W. and Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series, *Math. Comp.*, Vol. 19, pp. 297–301 (1965).

[6] Rader, C.M.: Discrete Fourier transforms when the number of data samples is prime, *Proc. IEEE (Letters)*, Vol. 56, pp. 1107–1108 (1968).

[7] Winograd, S.: On computing the DFT, *Math. Comp.*, Vol. 32, No. 1, pp. 175–199 (1978).

[8] Bergland, G.D.: A Fast Fourier Transform Algorithm Using 8 Iterations, *Math. Comp.*, Vol. 22, pp. 275–279 (1968).

[9] 日立製作所：HI-UX/MPP リモート DMA 転送 使用の手引 6A20-3-C21 (1996).