積和演算命令に向いた 8 基底 FFT カーネルの提案

高 橋 大 介[†,☆] 金 田 康 正[†]

本論文では，積和演算命令に向いた 8 基底 FFT カーネルを提案する．この 8 基底 FFT カーネルは積和演算命令を持つプロセッサにおいて，従来の 8 基底 FFT カーネルに比べて総演算命令数を削減する．提案した 8 基底 FFT カーネルを，積和演算命令を持つプロセッサを搭載したワークステーション IBM RS/6000 590 および共有メモリ型ベクトル並列計算機 NEC SX-4 に実現し，性能評価を行った．その結果，従来の 8 基底 FFT カーネルや，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルに比べても高い性能が得られた．

A New Radix-8 FFT Kernel Suitable for Multiply-add Instruction

Daisuke Takahashi[†,☆] and Yasumasa Kanada[†]

In this paper, we propose a new radix-8 fast Fourier transform (FFT) kernel suitable for the CPU with multiply-add instruction. The proposed radix-8 FFT kernel requires less floating-point instructions than does the conventional radix-8 FFT kernel on processors which have a multiply-add instruction. We implement this algorithm and evaluate its performance on the IBM RS/6000 590 workstation and NEC SX-4 shared-memory vector parallel computer both of which have a multiply-add instruction. The result shows that our radix-8 FFT kernel is faster than the conventional radix-8 FFT kernel or Goedecker's radix-4 FFT kernel.

1. はじめに

高速 Fourier 変換（fast Fourier transform，以下 FFT）[1)] は，科学技術計算において今日広く用いられているアルゴリズムである．

$n = 2^m$ 点の FFT を計算するにあたって，これまでに 2 基底の FFT [1)] や 4 基底[2)]，8 基底[2)] の FFT が提案されてきた．基底を大きくすることにより，演算回数，特に実数の乗算回数が減ることが知られている[2)~4)]．

最初に FFT が提案された 1960 年代，浮動小数点加算は浮動小数点乗算に比べてずっと高速であった．したがって，FFT においては実数の乗算回数を減らすようなアルゴリズムが多く提案されてきた[5)~8)]．

しかし今日では，多くのプロセッサにおいて浮動小数点加算と浮動小数点乗算は同じ速さで実行できる．さらに，加算と乗算を同時に行える積和演算命令を持つプロセッサも多い．

積和演算命令を持たないプロセッサでは，実数の加算回数と乗算回数の和が演算命令数となるが，積和演算命令を持つプロセッサでは，加算回数と乗算回数の比によって，演算命令数が変化する．

FFT において積和演算命令に着目した研究としては，Goedecker による 2，3，4，5 基底 FFT カーネルにおける積和演算命令に向いた手法[9)] が知られている．ところが，同様の手法を用いた 8 基底 FFT カーネルは提案されていない．その理由は，Goedecker による積和演算命令に向いた手法を 8 基底の場合に適用したとしても，4 基底の場合に比べて演算命令数が削減されないからであるとされている[9)]．

しかし，実際の FFT の性能は演算命令数だけではなく，ロードとストアの回数にも大きく影響される．

近年のプロセッサの演算速度に対するメモリのアクセス速度は相対的に遅くなってきており，メモリアクセス回数を減らすことは，より重要になっている．特に，SMP 構成の計算機では演算速度に対するメモリのアクセス速度の差は，さらに大きくなると予想される．

したがって，近年のプロセッサにおける FFT アルゴリズムは，演算回数だけではなく，メモリアクセス回数も減らすことが今まで以上に重要である．

8 基底の FFT は，2 基底や 4 基底の FFT と比べて

† 東京大学情報基盤センター
Information Technology Center, University of Tokyo
☆ 現在，埼玉大学大学院理工学研究科
Presently with Graduate School of Science and Engineering, Saitama University

演算回数が減るだけでなく，2 基底の FFT と比べてトータルのロードとストアの回数が 1/3 で済み，4 基底の FFT と比べても，ロードとストアの回数が 2/3 で済むという利点がある．これは，基底を大きくするに従ってデータを再利用できる回数が増えるためにロードとストアの回数が減るからである[4)]．

これらの事実から，積和演算命令を適用しやすい 8 基底 FFT カーネルを構築することにより，Goedecker による積和演算命令に向いた 2 基底や 4 基底の FFT カーネルに比べてもさらに高速に FFT が計算できると予想される．

本論文では積和演算命令に向いた 8 基底 FFT カーネルを提案するとともに，積和演算命令を持つプロセッサを搭載したワークステーション IBM RS/6000 590，および共有メモリ型ベクトル並列計算機 NEC SX-4 上に実現し，性能評価を行う．

なお特に断わらない限り，本論文で取り扱う FFT は複素 FFT を意味することとし，積和演算命令とは，$x = y + z * w$ のような，4 オペランドの積和演算命令を意味するものとする．ここで，x，y，z と w は浮動小数点レジスタである．また，本論文では積和演算命令を持つプロセッサにおいて，加算，乗算または積和演算はそれぞれ 1 命令で実行でき，実行に必要なマシンサイクル数は同じであると仮定する．さらに，実行された浮動小数点演算命令の数を,「演算命令数」と定義する．

以下，2 章で高速 Fourier 変換について，3 章で Goedecker の積和演算命令に向いた手法について述べる．4 章で従来の 8 基底 FFT カーネルについて述べ，5 章で本論文で提案する 8 基底 FFT カーネルを示す．6 章で演算命令数およびロードとストアの回数の比較を行い，7 章で誤差の評価を行う．8 章で性能評価結果を示す．最後の 9 章はまとめである．

2. 高速 Fourier 変換

FFT は，離散 Fourier 変換（discrete Fourier transform，以下 DFT）を高速に計算するアルゴリズムとして知られている．本論文では $\omega_n = e^{-2\pi i/n}$，$i = \sqrt{-1}$ とする．すると，DFT は次式で定義される．

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad 0 \le k \le n-1 \tag{1}$$

FFT カーネル[3),4)] は FFT において，最内側のループで計算される処理であり，FFT カーネルの基底（radix）を p で表すと，次式で表される．

$$Y(k) = \sum_{j=0}^{p-1} X(j)\Omega^j \omega_p^{jk} \tag{2}$$

ここで Ω はひねり係数（twiddle factor）[3)] と呼ばれる 1 の原始根であり，複素数である．

基底 p の FFT カーネルでは，入力データ $X(j)$ にひねり係数 Ω^j を掛けたものに対して p 点のショート DFT[10)] が実行される．

式 (2) を計算するために，今までにさまざまな手法が提案されている[11),12)]．

3. Goedecker の積和演算命令に向いた手法

Goedecker による積和演算命令に向いた手法[9)] を説明する．説明を簡単にするために，式 (2) において $p = 2$ の場合，つまり 2 基底 FFT カーネルを例に示す．以後，$X(j)$ の実数部，虚数部をそれぞれ $X_R(j)$，$X_I(j)$ とし，$Y(k)$ についても同様に $Y_R(k)$，$Y_I(k)$ とする．また，ひねり係数 Ω^j においても，実数部と虚数部をそれぞれ wr_j と wi_j とする．

従来の 2 基底 FFT カーネルは以下のように表される．

$$\begin{aligned}
&u0 = X_R(0)\\
&v0 = X_I(0)\\
&r = X_R(1)\\
&s = X_I(1)\\
&u1 = r * wr_1 - s * wi_1\\
&v1 = r * wi_1 + s * wr_1\\
&Y_R(0) = u0 + u1\\
&Y_I(0) = v0 + v1\\
&Y_R(1) = u0 - u1\\
&Y_I(1) = v0 - v1
\end{aligned}$$

この FFT カーネルを積和演算命令を持つプロセッサで実行する際には，$u1$ および $v1$ を計算するのに積和演算が合計 2 回と乗算が合計 2 回必要であり，$Y_R(0)$，$Y_I(0)$，$Y_R(1)$，$Y_I(1)$ の計算において加算が合計 4 回必要である．つまり，この FFT カーネルの総演算命令数は 8 回となる．なお，$u1$ や $v1$ が

$$\begin{aligned}
&u1 = 0 + r * wr_1\\
&u1 = u1 - s * wi_1\\
&v1 = 0 + r * wi_1\\
&v1 = v1 - s * wr_1
\end{aligned}$$

のように実行される場合においても，$u1$ および $v1$ を計算するのに積和演算が合計 4 回必要となり，$Y_R(0)$，$Y_I(0)$，$Y_R(1)$，$Y_I(1)$ の計算において加算が合計 4 回必要であるので，FFT カーネルの総演算命令数は 8 回となる．

1 $cos_4 = \cos(\pi/4)$
2 $u0 = X_R(0)$
3 $v0 = X_I(0)$
4 $r = X_R(1)$
5 $s = X_I(1)$
6 $u1 = r * wr_1 - s * wi_1$
7 $v1 = r * wi_1 + s * wr_1$
8 $r = X_R(2)$
9 $s = X_I(2)$
10 $u2 = r * wr_2 - s * wi_2$
11 $v2 = r * wi_2 + s * wr_2$
12 $r = X_R(3)$
13 $s = X_I(3)$
14 $u3 = r * wr_3 - s * wi_3$
15 $v3 = r * wi_3 + s * wr_3$
16 $r = X_R(4)$
17 $s = X_I(4)$
18 $u4 = r * wr_4 - s * wi_4$
19 $v4 = r * wi_4 + s * wr_4$
20 $r = X_R(5)$
21 $s = X_I(5)$
22 $u5 = r * wr_5 - s * wi_5$
23 $v5 = r * wi_5 + s * wr_5$
24 $r = X_R(6)$
25 $s = X_I(6)$
26 $u6 = r * wr_6 - s * wi_6$
27 $v6 = r * wi_6 + s * wr_6$
28 $r = X_R(7)$
29 $s = X_I(7)$
30 $u7 = r * wr_7 - s * wi_7$
31 $v7 = r * wi_7 + s * wr_7$
32 $r0 = u0 + u4$
33 $s0 = v0 + v4$
34 $r1 = u2 + u6$
35 $s1 = v2 + v6$
36 $r2 = u1 + u5$
37 $s2 = v1 + v5$
38 $r3 = u3 + u7$
39 $s3 = v3 + v7$
40 $r4 = u0 - u4$
41 $s4 = v0 - v4$
42 $r5 = u2 - u6$
43 $s5 = v2 - v6$
44 $r6 = u1 - u5$
45 $s6 = v1 - v5$
46 $r7 = u3 - u7$
47 $s7 = v3 - v7$
48 $u0 = r0 + r1$
49 $v0 = s0 + s1$
50 $u1 = r0 - r1$
51 $v1 = s0 - s1$
52 $u2 = r2 + r3$
53 $v2 = s2 + s3$
54 $u3 = r2 - r3$
55 $v3 = s2 - s3$
56 $Y_R(0) = u0 + u2$
57 $Y_I(0) = v0 + v2$
58 $Y_R(4) = u0 - u2$
59 $Y_I(4) = v0 - v2$
60 $Y_R(2) = u1 + v3$
61 $Y_I(2) = v1 - u3$
62 $Y_R(6) = u1 - v3$
63 $Y_I(6) = v1 + u3$
64 $u0 = r4 + s5$
65 $v0 = s4 - r5$
66 $u1 = r4 - s5$
67 $v1 = s4 + r5$
68 $u4 = r6 + s7$
69 $v4 = s6 - r7$
70 $u5 = s7 - r6$
71 $v5 = s6 + r7$
72 $u2 = cos_4 * (u4 + v4)$
73 $v2 = cos_4 * (v4 - u4)$
74 $u3 = cos_4 * (u5 + v5)$
75 $v3 = cos_4 * (u5 - v5)$
76 $Y_R(1) = u0 + u2$
77 $Y_I(1) = v0 + v2$
78 $Y_R(5) = u0 - u2$
79 $Y_I(5) = v0 - v2$
80 $Y_R(3) = u1 + u3$
81 $Y_I(3) = v1 + u3$
82 $Y_R(7) = u1 - u3$
83 $Y_I(7) = v1 - v3$

図 1 従来の 8 基底 FFT カーネル
Fig. 1 Conventional radix-8 FFT kernel.

Goedecker の積和演算命令に向いた手法では，$a \neq 0$ のときに

$$ax + by \rightarrow a(x + (b/a)y) \qquad (3)$$

の変形が可能であることを利用し，積和演算命令を持つプロセッサにおいて FFT カーネルの演算命令数を削減している．

従来の 2 基底 FFT カーネルでは $wr_1 \neq 0$ であるので，式 (3) の変形が可能であり，Goedecker による積和演算命令に向いた 2 基底 FFT カーネルは次のようになる．

$$wi_1 = wi_1/wr_1$$
$$u0 = X_R(0)$$
$$v0 = X_I(0)$$
$$r = X_R(1)$$
$$s = X_I(1)$$
$$u1 = r - s * wi_1$$
$$v1 = r * wi_1 + s$$
$$Y_R(0) = u0 + u1 * wr_1$$
$$Y_I(0) = v0 + v1 * wr_1$$
$$Y_R(1) = u0 - u1 * wr_1$$
$$Y_I(1) = v0 - v1 * wr_1$$

この Goedecker による積和演算命令に向いた 2 基底 FFT カーネルを積和演算命令を持つプロセッサで実行する際には，u_1，v_1，$Y_R(0)$，$Y_I(0)$，$Y_R(1)$，$Y_I(1)$ を計算するのに積和演算命令が合計 6 回必要である．つまり，この FFT カーネルの総演算命令数は 6 回で済むことが分かる．なお，$wi_1 = wi_1/wr_1$ の値はあらかじめ計算しておくものとする．

4. 従来の 8 基底 FFT カーネル

従来の 8 基底 FFT カーネル[2),4)] を**図 1** に示す．従来の 8 基底 FFT カーネルは，前半部分（行番号 1～31）と後半部分（行番号 32～83）に分けることができる．前半部分は入力データ $X(j)$ とひねり係数 Ω^j の積の計算であり，$j = 1, 2, \ldots, 7$ に対して行うので 7 回の複素数の乗算が必要になる．複素数の乗算は，通常の計算方法では実数の乗算が 4 回と実数の加算が 2 回必要になる．したがって，乗算と加算の比が 2:1 となり，積和演算命令を持つプロセッサでは，加算器が半分遊んでしまうことになる．

また，変換の後半部分は 4 回の実数の乗算に対して 52 回の実数の加算となっている．つまり，後半部分では逆に乗算器が有効に使われていないことが分かる．このように，従来の 8 基底 FFT カーネルは積和演算命令に適しているとはいいがたい．

5. 提案する 8 基底 FFT カーネル

4 章で述べたように，従来の 8 基底 FFT カーネルでは乗算と加算の回数がアンバランスであり，積和演算命令が活用できない．そこで，8 基底 FFT カーネルを変形して，積和演算命令を有効に活用することを考える．

式 (3) の変形を図 1 の従来の 8 基底 FFT カーネルに繰り返し適用することで，**図 2** に示すような積和

$cos_4 = \cos(\pi/4)$
$wi_1 = wi_1/wr_1$
$wi_2 = wi_2/wr_2$
$wi_3 = wi_3/wr_3$
$wi_4 = wi_4/wr_4$
$wi_5 = wi_5/wr_5$
$wi_6 = wi_6/wr_6$
$wi_7 = wi_7/wr_7$
$wr_{31} = wr_3/wr_1$
$wr_{51} = wr_5/wr_1$
$wr_{62} = wr_6/wr_2$
$wr_{73} = wr_7/wr_3$
$wr_{121} = wr_1 * cos_4$
$u0 = X_R(0)$
$v0 = X_I(0)$
$r = X_R(1)$
$s = X_I(1)$
$u1 = r - s * wi_1$
$v1 = r * wi_1 + s$
$r = X_R(2)$
$s = X_I(2)$
$u2 = r - s * wi_2$
$v2 = r * wi_2 + s$
$r = X_R(3)$
$s = X_I(3)$
$u3 = r - s * wi_3$
$v3 = r * wi_3 + s$
$r = X_R(4)$
$s = X_I(4)$
$u4 = r - s * wi_4$
$v4 = r * wi_4 + s$
$r = X_R(5)$
$s = X_I(5)$
$u5 = r - s * wi_5$
$v5 = r * wi_5 + s$
$r = X_R(6)$
$s = X_I(6)$
$u6 = r - s * wi_6$
$v6 = r * wi_6 + s$
$r = X_R(7)$
$s = X_I(7)$
$u7 = r - s * wi_7$
$v7 = r * wi_7 + s$
$r0 = u0 + u4 * wr_4$
$s0 = v0 + v4 * wr_4$
$r1 = u2 + u6 * wr_{62}$
$s1 = v2 + v6 * wr_{62}$
$r2 = u1 + u5 * wr_{51}$
$s2 = v1 + v5 * wr_{51}$
$r3 = u3 + u7 * wr_{73}$
$s3 = v3 + v7 * wr_{73}$
$r4 = u0 - u4 * wr_4$
$s4 = v0 - v4 * wr_4$
$r5 = u2 - u6 * wr_{62}$
$s5 = v2 - v6 * wr_{62}$
$r6 = u1 - u5 * wr_{51}$
$s6 = v1 - v5 * wr_{51}$
$r7 = u3 - u7 * wr_{73}$
$s7 = v3 - v7 * wr_{73}$
$u0 = r0 + r1 * wr_2$
$v0 = s0 + s1 * wr_2$
$u1 = r0 - r1 * wr_2$
$v1 = s0 - s1 * wr_2$
$u2 = r2 + r3 * wr_{31}$
$v2 = s2 + s3 * wr_{31}$
$u3 = r2 - r3 * wr_{31}$
$v3 = s2 - s3 * wr_{31}$
$Y_R(0) = u0 + u2 * wr_1$
$Y_I(0) = v0 + v2 * wr_1$
$Y_R(4) = u0 - u2 * wr_1$
$Y_I(4) = v0 - v2 * wr_1$
$Y_R(2) = u1 + v3 * wr_1$
$Y_I(2) = v1 - u3 * wr_1$
$Y_R(6) = u1 - v3 * wr_1$
$Y_I(6) = v1 + u3 * wr_1$
$u0 = r4 + s5 * wr_2$
$v0 = s4 - r5 * wr_2$
$u1 = r4 - s5 * wr_2$
$v1 = s4 + r5 * wr_2$
$u4 = r6 + s7 * wr_{31}$
$v4 = s6 - r7 * wr_{31}$
$u5 = s7 * wr_{31} - r6$
$v5 = s6 + r7 * wr_{31}$
$u2 = u4 + v4$
$v2 = v4 - u4$
$u3 = u5 + v5$
$v3 = u5 - v5$
$Y_R(1) = u0 + u2 * wr_{121}$
$Y_I(1) = v0 + v2 * wr_{121}$
$Y_R(5) = u0 - u2 * wr_{121}$
$Y_I(5) = v0 - v2 * wr_{121}$
$Y_R(3) = u1 + u3 * wr_{121}$
$Y_I(3) = v1 + v3 * wr_{121}$
$Y_R(7) = u1 - u3 * wr_{121}$
$Y_I(7) = v1 - v3 * wr_{121}$

図 2　提案する 8 基底 FFT カーネル
Fig. 2　New radix-8 FFT kernel.

演算命令に向いた 8 基底 FFT カーネルが導かれる．

なお，これらの変形はくくり出す変数が 0 でないことが明らかでないと行えず，最適化コンパイラでは困難な変形であることに注意しておく．

図 2 に示すように，提案した 8 基底 FFT カーネルでは，wi_1 から wi_7 の値は $\sin\alpha$ ではなく $\tan\alpha$ の形になっており，wr_{31} から wr_{73} の値は $\cos\alpha/\cos\beta$ の形になっていることが分かる．これらの値はあらかじめテーブルとして作成しておけるので，何回も FFT を実行する場合においてはオーバヘッドは無視できる．

6. 演算命令数およびロードとストアの回数の比較

演算命令数およびロードとストアの回数の比較にあたっては，従来の 4 基底 FFT カーネル，Goedecker による積和演算命令に向いた 4 基底 FFT カーネル[9]，従来の 8 基底 FFT カーネル，そして提案した 8 基底 FFT カーネルの 4 種類の各 FFT カーネルの浮動小数点演算命令数およびロードとストアの回数を比較した．議論の前提として，FFT カーネルが最も高速に実行できる場合とは，浮動小数点演算命令数およびロード＋ストア回数が最も少ない場合であるとする．したがって，浮動小数点演算命令数が同じであっても，ロード＋ストア回数が少なければ，その FFT カーネルは最も高速に実行できることになる．

表 1　積和演算命令を持つプロセッサにおける各 FFT カーネル内の演算命令数
Table 1　Number of floating-point instructions for FFT kernels on a processor with multiply-add instruction.

	浮動小数点演算	ロード	ストア
Conventional Radix-4	28	8	8
Goedecker Radix-4	22	8	8
Conventional Radix-8	84	16	16
New Radix-8	66	16	16

6.1 積和演算命令を持つプロセッサの場合

積和演算命令を持つプロセッサにおける，各 FFT カーネル内の演算命令数を**表 1** に示す．

表 1 において，浮動小数点演算と書いてある項目では，実数の乗算，加算，積和演算をそれぞれ 1 演算とした場合の合計の演算命令数を比較している．表 1 から分かるように，提案した 8 基底 FFT カーネルは，従来の 8 基底 FFT カーネルに比べて演算命令数が 84 回から 66 回に削減されている．これは約 21%の演算命令数の削減になる．

$n = p^t$ と表される場合，n 点 FFT の演算回数 T は，p 基底 FFT カーネルの演算回数を T_p とすると，

$$T = T_p \cdot \frac{1}{p} \log_p n = T_p \cdot \frac{t}{p} \tag{4}$$

表 2 積和演算命令を持つプロセッサにおける各 FFT カーネルによる n 点 FFT の演算命令数
Table 2 Number of floating-point instructions for n point FFTs on a processor with multiply-add instruction.

	浮動小数点演算	ロード ＋ ストア	比
Conventional Radix-4	$3.5n \log_2 n$	$2n \log_2 n$	1.75
Goedecker Radix-4	$2.75n \log_2 n$	$2n \log_2 n$	1.375
Conventional Radix-8	$3.5n \log_2 n$	$1.333n \log_2 n$	2.625
New Radix-8	$2.75n \log_2 n$	$1.333n \log_2 n$	2.063

表 3 積和演算命令を持たないプロセッサにおける各 FFT カーネル内の演算命令数
Table 3 Number of floating-point instructions for FFT kernels on a processor without multiply-add instruction.

	乗算	加算	ロード	ストア
Conventional Radix-4	12	22	8	8
Goedecker Radix-4	14	22	8	8
Conventional Radix-8	32	66	16	16
New Radix-8	38	66	16	16

表 4 積和演算命令を持たないプロセッサにおける各 FFT カーネルによる n 点 FFT の演算命令数
Table 4 Number of floating-point instructions for n point FFTs on a processor without multiply-add instruction.

	乗算 ＋ 加算	ロード ＋ ストア	比
Conventional Radix-4	$4.25n \log_2 n$	$2n \log_2 n$	2.125
Goedecker Radix-4	$4.5n \log_2 n$	$2n \log_2 n$	2.25
Conventional Radix-8	$4.083n \log_2 n$	$1.333n \log_2 n$	3.063
New Radix-8	$4.333n \log_2 n$	$1.333n \log_2 n$	3.25

で表される[4)].

表 1 に基づき式 (4) より算出した，積和演算命令を持つプロセッサにおける各 FFT カーネルによる n 点 FFT の演算命令数を**表 2** に示す．なお，表 2 において「比」とは，浮動小数点演算命令数をロード＋ストアの回数で割った値である．

表 2 から分かるように，提案した 8 基底 FFT カーネルの演算命令数は従来の 8 基底 FFT カーネルに比べて削減されているものの，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルと演算命令数は同一になっている．

ところが，提案した 8 基底 FFT カーネルのロード＋ストアの回数は，Goedecker の 4 基底 FFT カーネルの 2/3 になっている．つまり，浮動小数点演算とロード＋ストアの回数の比は Goedecker の 4 基底 FFT カーネルが 1.375 対 1 であるのに対して，提案した 8 基底 FFT カーネルでは 2.063 対 1 となっている．

したがって，提案した 8 基底 FFT カーネルは Goedecker の 4 基底 FFT カーネルに比べてメモリアクセスが少なく，有利であることが分かる．

6.2 積和演算命令を持たないプロセッサの場合

積和演算命令を持たないプロセッサにおける各 FFT カーネルの演算命令数を**表 3** に示す．

積和演算命令を持たないプロセッサの場合，提案した 8 基底 FFT カーネルの乗算命令数は，図 2 をそのまま計算すれば 62 回必要である．ところが，それらのうち共通な乗算が 24 回あるために，これらをあらかじめ計算しておくことにより，積和演算命令を持たないプロセッサの場合，乗算命令数は表 3 に示すように 38 回で済むことが分かる．

表 3 に基づき式 (4) より算出した，積和演算命令を持たないプロセッサにおける各 FFT カーネルによる n 点 FFT の演算命令数を**表 4** に示す．なお，表 4 において「比」とは，乗算+加算の回数をロード＋ストアの回数で割った値である．

表 4 から分かるように，提案した 8 基底 FFT カーネルの乗算と加算の合計演算命令数は Goedecker の 4 基底 FFT カーネルに比べて，約 4%の演算命令数の削減になっているが，従来の 8 基底 FFT カーネルに比べると演算命令数が約 6%増えている．これは，提案した 8 基底 FFT カーネルでは，乗算と加算のバランスを改善するために，乗算を増やしているからである．

したがって，積和演算命令を持たないプロセッサでは，提案した 8 基底 FFT カーネルは従来の 8 基底

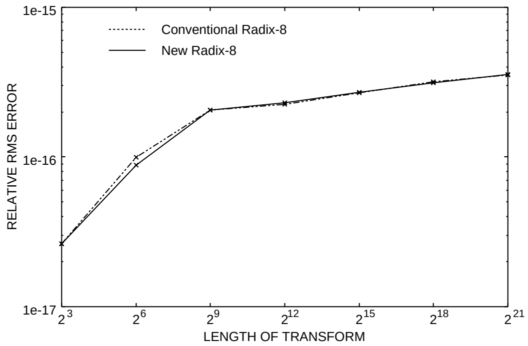

図 3　8 基底 FFT カーネルの相対 RMS 誤差（IBM RS/6000 590）
Fig. 3　Relative RMS error of radix-8 FFTs (IBM RS/6000 590).

FFT カーネルに比べて不利であることが分かる．

7. 誤差の評価

FFT における誤差は，FFT の実装方法やひねり係数の精度に依存することが知られている[13)~15)]．

誤差の評価にあたっては，文献 16) で行われているように，従来の 8 基底 FFT カーネルおよび提案した 8 基底 FFT カーネルの各々について，疑似乱数データに対する FFT と逆 FFT を IEEE 表現の倍精度計算で順に行った結果と，FFT を計算する前のデータを比較することにより行った．

図 3 に従来の 8 基底 FFT カーネル（図 1）の相対 RMS（Root Mean Square）誤差と提案した 8 基底 FFT カーネル（図 2）の相対 RMS 誤差を，積和演算命令を持つプロセッサを搭載したワークステーション IBM RS/6000 590 において比較したものを示す．コンパイラは IBM の XL Fortran V3.2 を用い，最適化オプションとして`-O3 -qarch=pwr2 -qtune=pwr2 -qstrict`を指定した☆．

図 3 から分かるように，提案した 8 基底 FFT カーネルは従来の 8 基底 FFT カーネルに比べて $2^6\,(=64)$ 点 FFT において若干誤差が増えているものの，精度の大きな低下は見られない．誤差が増える原因としては，提案した 8 基底 FFT カーネルを積和演算命令を持つプロセッサで実行する場合，従来の 8 基底 FFT カーネルに比べて浮動小数点演算の演算命令数は少なくなっているが，加減算回数に変化はないものの乗算回数が 32 回から 62 回に増加していることによるものと推察される．

8. 性能評価

性能評価にあたっては，FFT カーネルのみの性能を比較するために，複数（m）組の n 点 FFT を同時に計算し，実行時間を比較した．このような最内側ループが m 回繰り返される実行形態は，“four step” FFT [4),17)] や多次元 FFT に見ることができる．MFLOPS 値および GFLOPS 値の算出にあたっては，今回取り扱っている FFT が複素 FFT であるので，n 点 FFT の演算回数を $5n\log_2 n$ とし，m 組の n 点 FFT の演算回数は $5m\cdot n\log_2 n$ とした．

なお FFT の計算は倍精度複素数で行い，三角関数のテーブルはあらかじめ作り置きとしている．

計算機としては，ワークステーション IBM RS/6000 590（POWER2 66 MHz，ピーク性能 266 MFLOPS）および共有メモリ型ベクトル並列計算機 NEC SX-4（1 CPU あたりのピーク性能 2 GFLOPS）を用いた．これら 2 つの計算機には，積和演算命令を持つプロセッサが搭載されている．

8.1 IBM RS/6000 590 による測定結果

IBM RS/6000 590 においては，IBM のライブラリである ESSL V2.2 の FFT ルーチン（`DCFT`）および，Goedecker による積和演算命令に向いた 4 基底 FFT カーネル，従来の 8 基底 FFT カーネル，そして提案した 8 基底 FFT カーネルの 4 種類の性能の比較を行った．

コンパイラは IBM の XL Fortran V3.2 を用い，最適化オプションとして`-O3 -qarch=pwr2 -qhot -qtune=pwr2`を指定した．測定に際しては，CPU 時間を測定した．IBM RS/6000 590 における 64 組の $64\,(=4^3=8^2)$ 点 FFT および 1024 組の 64 点 FFT の実行時間を**表 5** に示す．

表 5 から分かるように，64 組の 64 点 FFT でも 1024 組の 64 点 FFT においても，提案した 8 基底 FFT カーネルは，ESSL の `DCFT` ルーチン，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルや従来の 8 基底 FFT カーネルに比べて，高い性能が得られている．

まず，提案した 8 基底 FFT カーネルと，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルの性能について考察する．今回評価に用いた IBM RS/6000 590 ではデータキャッシュの大きさが 128 KB であり，搭載されている POWER2 プロセッサは浮動小数点演算命令とロード＋ストア命令をオーバラップして実行できる．さらに表 2 から分かるように，4，8 基底

☆ “`-O3`” は最適化のレベルが 3 を意味し，“`-qarch=pwr2`”，“`-qtune=pwr2`” は POWER2 向けの最適化を意味する．“`-qstrict`” は，“`-O3`” が行う最適化のうちプログラムの意味が変わらない最適化のみを行うオプションである．

表 5 4，8 基底の FFT カーネル（64 および 1024 組の 64 点 FFT）の性能（IBM RS/6000 590）
Table 5 Performance of radix-4, 8 FFT kernel (execution times of 64 and 1024 evaluations of 64-point FFTs) on IBM RS/6000 590.

	64 組の 64 点 FFT		1024 組の 64 点 FFT	
	Time (msec)	MFLOPS	Time (msec)	MFLOPS
DCFT (ESSL)	0.5157	238.26	17.139	114.71
Goedecker Radix-4	0.4913	250.10	16.094	122.16
Conventional Radix-8	0.5304	231.68	12.031	163.41
New Radix-8	0.4865	252.61	11.465	171.49

表 6 4，8 基底の FFT カーネル（65536 組の 64 点 FFT）の性能（NEC SX-4）
Table 6 Performance of radix-4, 8 FFT kernel (execution times of 65536 evaluations of 64-point FFTs) on NEC SX-4.

# CPU	Conventional Radix-4		Goedecker Radix-4		Conventional Radix-8		New Radix-8	
	Time (sec)	GFLOPS	Time (sec)	GFLOPS	Time (sec)	GFLOPS	Time (sec)	GFLOPS
1	0.07844	1.604	0.06931	1.815	0.07458	1.687	0.06817	1.846
2	0.03942	3.192	0.03479	3.617	0.03750	3.355	0.03410	3.690
4	0.01974	6.373	0.01745	7.209	0.01873	6.717	0.01710	7.359

表 7 4，8 基底の FFT カーネル（1024 組の 4096 点 FFT）の性能（NEC SX-4）
Table 7 Performance of radix-4, 8 FFT kernel (execution times of 1024 evaluations of 4096-point FFTs) on NEC SX-4.

# CPU	Conventional Radix-4		Goedecker Radix-4		Conventional Radix-8		New Radix-8	
	Time (sec)	GFLOPS	Time (sec)	GFLOPS	Time (sec)	GFLOPS	Time (sec)	GFLOPS
1	0.16720	1.505	0.14463	1.740	0.16147	1.559	0.14119	1.782
2	0.08399	2.996	0.07239	3.476	0.08054	3.125	0.07067	3.561
4	0.04205	5.985	0.03628	6.936	0.04044	6.224	0.03548	7.092

FFT カーネルでは，浮動小数点演算命令数に比べてロード＋ストア命令が少なくなっている．

64 組の 64 点 FFT ではデータの大きさが 64 KB となり，キャッシュにデータが入ってしまうために，ロード＋ストア命令の実行サイクルは浮動小数点演算命令の実行サイクルにほとんど隠れてしまう．したがって，この場合においては性能を決定する主な要因は浮動小数点演算命令数となる．Goedecker による積和演算命令に向いた 4 基底 FFT カーネルと提案した 8 基底 FFT カーネルでは浮動小数点演算命令数が同じであるために，性能は約 1%しか向上していない．

ところが，1024 組の 64 点 FFT ではデータの大きさが 1 MB になり，キャッシュにデータが入り切らないために，ロード＋ストア命令の実行サイクルは浮動小数点演算命令の実行サイクルで隠すことができなくなる．したがって，この場合においては性能を決定する主な要因は浮動小数点演算命令数ではなく，ロード＋ストア回数となる．

このように，ロード＋ストア命令の実行サイクルが浮動小数点演算命令の実行サイクルで隠せない場合においては，提案した 8 基底 FFT カーネルは Goedecker による積和演算命令に向いた 4 基底 FFT カーネルに比べてロード＋ストア回数が 2/3 であるため，理想的には 50%高速になると考えられる．実際の性能評価の結果は，表 5 より提案した 8 基底 FFT カーネルでは 171.49 MFLOPS，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルでは 122.16 MFLOPS であるので，提案した 8 基底 FFT カーネルが約 40%高速になっている．

この結果より，提案した 8 基底 FFT カーネルは，ロード＋ストア命令の実行サイクルが浮動小数点演算命令の実行サイクルで隠せない場合については，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルに比べて有利であるといえる．

次に，提案した 8 基底 FFT カーネルと，従来の 8 基底 FFT カーネルの性能について考察する．表 2 から分かるように，提案した 8 基底 FFT カーネルは従来の 8 基底 FFT カーネルに比べて演算命令数が約 21%削減されているが，表 5 の 64 組の 64 点 FFT では約 9%の高速化にとどまっている．原因としては，提案した 8 基底 FFT カーネルと従来の 8 基底 FFT カーネルではロード＋ストア回数が同一であること，そしてロード＋ストア命令の実行サイクルが浮動小数点演算命令の実行サイクルに完全には隠れていないために，浮動小数点演算命令数の比のとおりには高速化されていないことが考えられる．

8.2 NEC SX-4 による測定結果

NEC SX-4 においては，IBM RS/6000 590 の場

合と同様に，従来の 4 基底 FFT カーネルおよび，Goedecker による積和演算命令に向いた 4 基底 FFT カーネル，従来の 8 基底 FFT カーネル，そして提案した 8 基底 FFT カーネルの 4 種類の性能の比較を行った．

NEC SX-4 の評価に際しては，1 CPU〜4 CPU による経過時間を測定した．コンパイラは NEC の FORTRAN77/SX（Rev. 134）を用い，最適化オプションとして 1 CPU の実行に際しては `-C hopt -pi` を用い，2 CPU，4 CPU の実行に際しては `-P auto -C hopt -pi` を用いた☆．なお 2 CPU，4 CPU による実行では，最内側ループの部分が並列実行されている．NEC SX-4 における 65536 組の 64 $(= 4^3 = 8^2)$ 点 FFT および 1024 組の 4096 $(= 4^6 = 8^4)$ 点 FFT の実行時間を**表 6**，**表 7** に示す．

まず，提案した 8 基底 FFT カーネルと，Goedecker による積和演算命令に向いた 4 基底 FFT カーネルの性能について考察する．表 6，表 7 によると，提案した 8 基底 FFT カーネルは Goedecker による積和演算命令に向いた 4 基底 FFT カーネルに比べて約 1.7%〜2.4%高速になっている．

今回評価に用いた NEC SX-4 では各 CPU でベクトル処理を行っており，データ数が大きくなってもメモリアクセス性能は低下していない．さらに，NEC SX-4 では POWER2 プロセッサと同様に浮動小数点演算命令とロード＋ストア命令をオーバラップして実行できること，そして 4 CPU の場合でもメモリバンド幅に余裕があるために，ロード＋ストア命令の実行サイクルは浮動小数点演算命令の実行サイクルに隠れる場合が多くなる．つまり，IBM RS/6000 590 においてキャッシュにデータが入ってしまう場合と同様な傾向になると考えられる．

しかし，ロード＋ストア命令の実行サイクルが浮動小数点演算命令の実行サイクルに完全には隠れるわけではないので，提案した 8 基底 FFT カーネルは Goedecker による積和演算命令に向いた 4 基底 FFT カーネルに比べてロード＋ストア回数が少ない分，高い性能が得られていることが分かる．

次に，提案した 8 基底 FFT カーネルと，従来の 8 基底 FFT カーネルの性能について考察する．表 6，表 7 から分かるように，提案した 8 基底 FFT カーネルは従来の 8 基底 FFT カーネルに比べて演算命令数が約 21%削減されているが，約 9%〜14%の高速化にとどまっている．原因としては，IBM RS/6000 590 の場合と同様に，提案した 8 基底 FFT カーネルと従来の 8 基底 FFT カーネルではロード＋ストア回数が同一であること，そしてロード＋ストア命令の実行サイクルが浮動小数点演算命令の実行サイクルに完全には隠れていないために，浮動小数点演算命令数の比のとおりには高速化されていないことが考えられる．

☆ “`-C hopt`” は最適化およびベクトル化機能を最大限に利用できる翻訳モードを指定するオプションであり，“`-pi`” は最適化プリプロセッサを使用した手続きのインライン展開機能を利用することを指定するオプションである．また，“`-P auto`” は最適化プリプロセッサを使用した自動並列化機能を利用することを指定するオプションである．

9. ま と め

本論文では，積和演算命令に向いた 8 基底 FFT カーネルを提案するとともに，実際に評価を行った．Goedecker による積和演算命令に向いた手法を 8 基底の場合に適用しても，4 基底の場合に比べて演算命令数が削減されないことが知られているが，8 基底 FFT のロードとストア回数は 4 基底 FFT に比べて 2/3 で済むことに着目し，従来の 8 基底 FFT カーネルを積和演算命令に向いた形に変形した．その結果，積和演算命令を持つプロセッサにおいて，従来の 8 基底 FFT カーネルや Goedecker による積和演算命令に向いた 4 基底 FFT カーネルに比べても高い性能が得られることを確認した．

提案した 8 基底 FFT カーネルは，積和演算命令を持つプロセッサにおいて，乗算回数は増えるが演算命令数は減るという，一見相反する変形を行うとともに，総演算命令数に対するメモリへのロード・ストア総数比も減少させることで高速化を図っている．今回適用した手法は，他の基底，たとえば 6 基底[8),18)]，12 基底[18)]，16 基底[2)] の FFT カーネルなどにも有効に適用できると考えられる．これらの実現と評価は今後の課題である．

謝辞　有益なコメントをくださった査読者の方々に感謝いたします．なお本研究の一部は，文部省科学研究費補助金奨励研究（A）（課題番号 10780166）の支援を受けた．

参 考 文 献

1) Cooley, J.W. and Tukey, J.W.: An Algorithm for the Machine Calculation of Complex Fourier Series, *Math. Comput.*, Vol.19, pp.297–301 (1965).

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

3) Brigham, E.O.: *The Fast Fourier Transform*

and its Applications, Prentice-Hall, Englewood Cliffs, NJ (1988).
4) Van Loan, C.: *Computational Frameworks for the Fast Fourier Transform*, SIAM Press, Philadelphia, PA (1992).
5) Kolba, D.P. and Parks, T.W.: A Prime Factor FFT Algorithm Using High-Speed Convolution, *IEEE Trans. Acoust. Speech Signal Process.*, Vol.ASSP-25, pp.281–294 (1977).
6) Winograd, S.: On Computing the Discrete Fourier Transform, *Math. Comput.*, Vol.32, pp.175–199 (1978).
7) Dubois, E. and Venetsanopoulos, A.: A New Algorithm for the Radix-3 FFT, *IEEE Trans. Acoust. Speech Signal Process.*, Vol.ASSP-26, pp.222–225 (1978).
8) Prakash, S. and Rao, V.V.: A New Radix-6 FFT Algorithm, *IEEE Trans. Acoust. Speech Signal Process.*, Vol.ASSP-29, pp.939–941 (1981).
9) Goedecker, S.: Fast Radix $2, 3, 4$, and 5 Kernels for Fast Fourier Transformations on Computers with Overlapping Multiply-Add Instructions, *SIAM J. Sci. Comput.*, Vol.18, pp.1605–1611 (1997).
10) Nussbaumer, H.J.: *Fast Fourier Transform and Convolution Algorithms*, second corrected and updated edition, Springer-Verlag, New York (1982).
11) Singleton, R.C.: An Algorithm for Computing the Mixed Radix Fast Fourier Transform, *IEEE Trans. Audio Electroacoust.*, Vol.17, pp.93–103 (1969).
12) Temperton, C.: Self-Sorting Mixed-Radix Fast Fourier Transforms, *J. Comput. Phys.*, Vol.52, pp.1–23 (1983).
13) Kaneko, T. and Liu, B.: Accumulation of Round-off Error in Fast Fourier Transforms, *J. ACM*, Vol.17, pp.637–654 (1970).
14) Calvetti, D.: A Stochastic Roundoff Error Analysis for the Fast Fourier Transform, *Math. Comput.*, Vol.56, pp.755–774 (1991).
15) Schatzman, J.C.: Accuracy of the Discrete Fourier Transform and the Fast Fourier Transform, *SIAM J. Sci. Comput.*, Vol.17, pp.1150–1166 (1996).
16) Bailey, D.H.: A High-Performance FFT Algorithm for Vector Supercomputers, *Int. J. Supercomputer Applications*, Vol.2, pp.82–87 (1988).
17) Bailey, D.H.: FFTs in External or Hierarchical Memory, *The J. Supercomputing*, Vol.4, pp.23–35 (1990).
18) Suzuki, Y., Sone, T. and Kido, K.: A New FFT Algorithm of Radix 3, 6, and 12, *IEEE Trans. Acoust. Speech Signal Process.*, Vol.ASSP-34, pp.380–383 (1986).

(平成 11 年 8 月 31 日受付)
(平成 12 年 5 月 11 日採録)

高橋　大介（正会員）

1970 年生．1991 年呉工業高等専門学校電気工学科卒業．1993 年豊橋技術科学大学工学部情報工学課程卒業．1995 年同大学大学院工学研究科情報工学専攻修士課程修了．1997 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．1999 年同大学情報基盤センター助手．2000 年埼玉大学大学院理工学研究科情報数理科学専攻助手．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．

金田　康正（正会員）

1949 年生．1973 年東北大学理学部物理第二学科卒業．1978 年東京大学大学院理学系研究科博士課程修了．理学博士．1978 年名古屋大学プラズマ研究所助手，1981 年東京大学大型計算機センター助教授，同教授を経て現在東京大学情報基盤センター教授．その間英国ケンブリッジ大学計算機研究所客員研究員，名古屋大学プラズマ研究所客員助教授，核融合科学研究所客員助教授．昭和 58 年度（欧文）および平成 10 年度（邦文）情報処理学会論文賞受賞．平成 6 年度情報処理学会 Best Author 賞受賞．著書「π のはなし」（東京図書），共著「アドバンスト・コンピューティング—21 世紀の科学技術基盤」（培風館），監訳「よくわかる計算機物理—コンピュータ・シミュレーションと計算機代数入門」，編著「Trends in Supercomputing」（World Scientific）．日本応用数理学会，プラズマ・核融合学会，ACM，SIAM 各会員．研究テーマは「大規模数値計算」，「知識発見」および「研究の研究」．