社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2004－SLDM－117（40）
2004／12／2

FFT乗算器の最適化実装

矢崎 俊志† 阿部 公輝†

† 電気通信大学 電気通信学研究科 〒182-8585 東京都調布市調布ヶ丘 1-5-1
E-mail: †{syunji,abe}@cacao.cs.uec.ac.jp

あらまし 暗号計算などで用いられる多倍長演算に,FFTを用いる方法がある. ここでは,FFT乗算器の最適なハードウェア実装について述べる. FFT乗算では, 要求される精度が計算桁数に依存するので, 浮動小数点数値表現の最小ビット数は, 計算桁数によって変わる. 本研究では, 数値実験による誤差解析を行い, その結果にもとづいて最適なデータ表現を用いたFFT乗算器の実装を行った. 結果,2^{11}桁どうしの乗算において,IEEE754 64bit表現を用いた場合と比較して, 面積を1/3に削減し, 速度を1.3倍向上させることができた.

キーワード FFT, 乗算, 多倍長,VLSI, データ表現

An Optimum Implementation of FFT Multiplier

Syunji YAZAKI† and Kôki ABE†

† Department of Computer Science, The University of Electro-Communications 1-5-1 Chofugaoka Chofu-shi,Tokyo 182-8585 Japan
E-mail: †{syunji,abe}@cacao.cs.uec.ac.jp

Abstract FFT can be useful to calculate the multi-digit multiplication used for cryptography. Here, we describe an optimum hardware implementation of an FFT multiplier. Because the required precision depends on the number of digits to be calculated, the least bit length of the floating point number varies. We performed an experimental error analysis and implemented an FFT multiplier using an optimum data representation obtained by the analysis. We found from the implementation that our optimized FFT multiplier could reduce the area by 1/3 and improve the performance 1.3 times, compared to the 64-bit IEEE754 representation in multiplying hexadecimal values of 2^{11} digits.

Key words FFT,multi-digit,multiplication,hardware,VLSI,data representation

1. はじめに

多倍長演算は, 数値演算で精度の良い近似解を得ることなどに使われてきたが, 近年, 暗号等への応用が期待されている. 文献[1]では, 公開鍵暗号系で重要となる大きい桁数の素数判定を決定的多項式時間で行うアルゴリズムが紹介されているが, このアルゴリズムにFFTを利用した多倍長乗算を用いることで, 計算量を改善することができると報告されている. 他にも, カオスの初期値敏感性を利用したカオス暗号や通信では, 解の有効桁数が信頼性に影響を与えることが知られており[2], 有効桁数を多く保つために, 多倍精度が用いられる例もある[3].

一般に, 多倍長演算の中でも応用範囲の広い乗算に関する研究は多数あり, Karatsuba法,Toom-Cook法,FFT乗算などが一般に知られている. また,Zurasは,2-way法(Karatsuba法),3-way法(Toom-Cook法),4-way法,5-way法, FFT乗算, それぞれの計算量を比較し, 桁数が大きい程,FFT乗算が他の方法より速度の面で有利であるとしている[4].

FFT乗算はその応用例から, 高速な動作が要求される. この乗算法は,FFTの計算量が全体の計算量の大半を占めるため, 乗算の高速化のためにはFFTの高速化が不可欠である.FFTの高速化に関しては, 効率のよいFFTアルゴリズムの考案[5],[6], 高速かつ低面積, 低消費電力のハードウェア実装[7],[8]などがなされている. しかし, これらの研究は,FFT単体の高速化を主としており,FFTを乗算に利用することを前提としたものではない.

FFT乗算は, 整数乗算に実数演算を用いるため, 演算中に誤差が発生, 拡大する. したがって,FFTを乗算に利用する場合, 最適化の指標として, 演算に必要な精度を考慮にいれなければならない. ソフトウェアでFFT乗算を実装する場合, 精度に関する選択肢は限られている. 一方, ハードウェアで実装する場合, データ表現は自由であるが, 過去に乗算への応用を視野にいれてFFTの最適化を行っている例は知られていない. そこで, 本

研究では,FFT 乗算を行う場合の誤差に着目し, 計算桁数が与えられたとき, その演算に最低限必要な精度を持つデータ長で演算器を構成し, 面積, 速度を最適化した FFT 乗算器を設計する.

本論文では, 第 2 章で,FFT 乗算とその他の多倍長乗算について比較する. 次に, 第 3 章で FFT 乗算と誤差の関連を述べ, 第 4 章で実装した FFT 乗算器の構成を説明する. さらに, 第 5 章で誤差の実験的解析について述べ, 計算桁数に対する最適なデータ長を求める. その結果をもとに第 6 章で実装を行い結果を考察する. 最後に, 第 7 章でまとめる.

2. FFT 乗算

FFT 乗算は, 畳み込みを利用して多倍長数の積を高速に求める乗算法である. 次にそのアルゴリズムを説明する.

r 進数 m 桁の整数 u, v を虚数部がすべて 0 の $2m$ 次元複素ベクトルで $u = (0 \ldots 0, a_{m-1} \ldots a_0)_b$,$v = (0 \ldots 0, b_{m-1} \ldots b_0)_b$ と表す. ただし,$2m = 2^k (k$ は非負整数) とする.u と v の積を $h = u \cdot v = (c_{2m-1} \ldots c_0)_b$ とし, 関数 $F(\cdot)$ を FFT 順変換, 関数 $F^{-1}(\cdot)$ を FFT 逆変換とすると,u,v,h の間に式 (1) の関係が成り立つ [9]. ここに,$\otimes$ はベクトルの要素ごとの乗算を表す.

$$h = F^{-1}\left(F(u) \otimes F(v)\right)/2m \qquad (1)$$

すなわち, 整数 m 桁の乗数と被乗数が与えられたとき, それぞれをすべての要素の虚数部が 0 の m 次元複素ベクトルとみなす. さらに, 上位を m 個の 0 でパディングし $2m$ 次元複素ベクトルを作り, それぞれに対して FFT を行う. 得られた 2 つの結果を要素ごとに乗算し,FFT 逆変換を行うと, 得られた $2m$ 次元複素ベクトルが積となっている. ただし, この時点の積は, 各桁が $2m$ 倍された複素数で表現されている. さらに, 桁上げも行われていない. そこで, 各桁を $1/2m$ し, 四捨五入によって整数に丸め, さらに基数 r として桁上げの処理を行うことで最終的な積を得ることができる.

FFT 乗算を用いた場合,m 桁どうしの乗算に必要な計算量は $O(m \log m)$ である. FFT 乗算以外にも乗算を高速に行う演算法は多く提案されており, それぞれの演算法の計算量は,Zuras によって見積もられている [4]. これには, 計算量の面で FFT 乗算が最も高速であることが述べられている. また, 実際の計算機上の実験においても,FFT の乗算法が最も高速であることが確認されている. ただし,FFT 乗算は他の演算法と比較して, 演算のオーバヘッドが大きいため,m が大きい場合に有効な方法である.

3. 誤　　差

FFT 乗算は, 実数の演算を必要とするため, 誤差が発生する. この誤差は, 演算中に拡大し, 最終的な結果を四捨五入で整数に丸める際の絶対誤差が 0.5 以上になっていると, 値が目的の整数の一つ隣りの整数に丸められてしまい, 正確な結果が得られない. よって,FFT 乗算を利用する場合は, 誤差には特に注意する必要がある.

FFT 乗算における誤差の見積もりは,Henrici によって解析的に行われており [10], マシンイプシロン ϵ_M のデータ表現を用いて, 桁数 m, 基数 r の FFT 乗算を行った場合, 正しい積を得るには, 次式を満足する必要があるとしている.

$$\epsilon_M \leqq \frac{1}{192m^2(2\log_2 m + 7)r^2} \qquad (2)$$

一方, 平山は,Henrici が行った解析的な誤差の見積もりは最悪ケースであり, 実際の計算では, 誤差の条件はもっと緩いことを指摘し, 数値実験により確認した [11].

文献 [11] の実験では,IEEE754 や IBM 形式など, 広く一般的に利用されている浮動小数点表現を用いて, 基数 r と桁数 m を変化させながら FFT 乗算を行い,IEEE754 や IBM 方式のデータ表現を用いた場合, 何進数何桁まで正しい結果を得ることができるのかを測定している. 測定にあたり r 進数 m 桁で表現できるオペランドが取り得るすべての値で誤差を測定し正しい積が得られることを確認することはできないので, これらのオペランドの中で, 最大の誤差が発生するもので演算を行い, その結果からすべての組合せの演算で正しい結果が得られることを保証している. この最大の誤差を与えるオペランドに関して, 平山は式 (2) を誤差の評価式とみなし, 乗算に要求される精度が, 基数 r が大きいほど高くなる点から, すべての桁が $r-1$ となるようなオペランドが最大の誤差を与えるとしている.IEEE754 の 64bit 表現を用いた場合, 仮数部長は 52bit であるので, マシンイプシロンは $\epsilon_M \fallingdotseq 2.220446 \times 10^{-16}$ である. これを式 (2) で評価した場合,10 進 77091 桁まで計算可能であるという結果が得られるのに対し, 平山の実験では 10 進数百万桁相当まで演算可能であることが示されている. 第 5 章では, 平山の実験的誤差解析に基づき, 与えられた計算桁数の多倍長乗算を行うために最適なデータ表現を求める.

4. FFT 乗算器の構成

第 2 章で述べたアルゴリズムで FFT 乗算を行うために必要な演算器の構成を図 1 に示す.

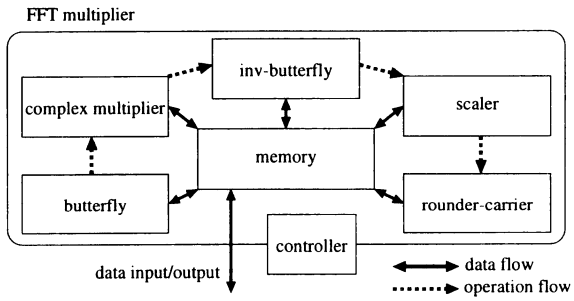

図 1　FFT 乗算器の構成

FFT 乗算器は, 順・逆 FFT の基本演算である順・逆バタフライ演算器 (butterfly,inv-butterfly), 複素数乗算器 (complex multiplier), 逆 FFT 後の値の調整を行うスケーラ (scaler), 四捨五入と桁上げを行うモジュール (rounder-carrier), 値を保存するメモリ (memory), 演算やデータの流れを制御するコントローラ (controller) から構成される. 図 1 中の破線矢印は, 演算

の流れを示し, 実線の矢印はデータの流れを示している. 次に, 各モジュールの詳細な説明を行う.

4.1 butterfly, inv-butterfly

butterfly と inv-butterfly は,FFT を行うための基本演算であるバタフライ演算を実現するモジュールである. 次元 $2m$ のベクトルに対する FFT では, このバタフライ演算が $2m/t \times \log_t 2m$ 回行われる. 対数の底 t は FFT の基数である.

モジュール butterfly は FFT の順変換に,inv-butterfly は逆変換に用いられる. バタフライ演算の内容は FFT の種類によって変わる. 選択可能な FFT のアルゴリズムには様々な種類があるが, 今回は演算器の単純化を考慮し, 最も基本的な,Cooley-Tukey 型 FFT(CT-FFT) [12] を用いる.CT-FFT を用いた場合,butterfly の演算は変換前の値を x,y, 変換後の値を X,Y とすると, 次式で表される.

$$X = x + yW \tag{3}$$

$$Y = y - yW \tag{4}$$

ここに,W は FFT における回転子を表し,1 の原始 $2m$ 乗根の 1 つである. どの原始根を演算に用いるかは, 過去に行われたバタフライ演算の回数から機械的に決定することができる.

また,inv-butterfly の演算は変換前の値を X',Y', 変換後の値を x',y' とすると, 次式で表される.

$$x' = X' + Y' \tag{5}$$

$$y' = (X' - Y')W \tag{6}$$

4.2 complex multiplier

複素数の乗算を行う complex multiplier モジュールは, $a + bi$,$c + di$ が与えられた時, 式 (7) にしたがって複素数の乗算を行う. ただし,i は虚数単位とする.

$$\begin{aligned}(a + bi)(c + di) &= ac - bd + (ad + bc)i \\ &= ac - bd + ((a + b)(c + d) - (ac + bd))i\end{aligned} \tag{7}$$

式 (7) の式変形により, 積 ac,bd をそれぞれ 1 回計算することにより, 乗算の数を 4 回から 3 回に削減している. 加算の回数は増加するが, 一般に, 加算より乗算の方が演算コストが多い.

4.3 scaler

式 (1) に示すように,$2m$ 次元ベクトルに対して FFT 順変換を行い, それを逆変換すると, すべての要素は $2m$ 倍される. これを調整するために, 各桁を $2m$ で除する必要がある.$2m = 2^k$(k は非負整数) の場合だけを考え, データ表現に仮数部の基数が 2 の浮動小数点表現を用いれば, 実際に除算を行う必要はなく, 浮動小数点で表現された値の指数部から k を減ずればよい. scaler は以上の演算を行うモジュールである.

FFT は対称性を持っているため,$2m$ 次元の実数ベクトルを順変換すると, $m+i$ 番目の要素と $m-i$ 番目の要素が複素共役になる (複素共役対称). また逆に, 複素共役対称ベクトルを逆変換すると, 実数ベクトルになる. 一方, 複素共役対称ベクトルどうしを要素ごとに掛けても複素共役対称性は維持される.FFT 乗算において, 乗数と被乗数は整数ベクトルであるので, これらを順変換すると対称性から複素共役対称ベクトルになる. さらに要素ごとの乗算をした後に逆変換をすると, 結果は実数ベクトルになるはずなので, 虚数部が 0 でなかったとしても, それは誤差によるものとして以後の過程においては無視することができる. したがって,scaler による操作は実数部に対してのみ行えばよい.

4.4 rounder-carrier

最終的な積を得るためには, 各桁が実数で表現された積の各桁を整数に丸める必要がある.

FFT 乗算の場合, 誤差による影響を最小にするためには, 最も近い整数に丸める必要がある. これによって, 絶対誤差が ± 0.5 を超えない限り正しい結果が得られる. その後, 各桁を基数 r にしたがって桁上げしてゆく. これらの操作を行うモジュールが rounder-carrier である.

4.5 memory

多倍長の演算を行う場合, 演算の対象となるデータを格納する記憶領域は巨大なものとなる. したがって, ハードウェアで多倍長演算を実装する場合, データ全体は外部の大規模な記憶装置に格納し, 演算はキャッシュ上で行う.memory モジュールはこのキャッシュとして機能する.

演算のパイプライン化を考えた場合, 同一のメモリに対する同時読み書きによって, 構造ハザードが発生する場合がある. これを回避するために,memory モジュールは Read,Write のポートを 1 個ずつ持つ Dual Port RAM を用いる.

4.6 controller

FFT 乗算器を構成する各演算器を制御するモジュールが controller である. 具体的には, 各演算器,memory からデータ入出力, 外部とのデータのやり取りを制御する.

5. 最適なデータ表現

第 3 章で述べたように,FFT 乗算を行う際には, 演算の精度に注意する必要がある. しかし, ソフトウェアを用いて数千から数万桁の FFT 乗算を行う場合は,64bit の浮動小数点表現等を用いることにより, 十分に高い精度で演算を行うことができるので, これらを厳密に意識する場面はあまりない. また, 実装コストの面でも, 低い精度を用いた場合と, 高い精度を用いた場合とではほとんどかわらない. しかし, ハードウェアで実装する場合には, 余剰な精度は面積の増加と速度の低下につながる. したがって, 最適な設計をするためには最低限必要な精度を求め, それに基づく最適なデータ表現を用いる必要がある.

実数のデータ表現は様々あるが, ソフトウェアを用いた場合, その選択肢は限られている. 一方, ハードウェア実装の場合, データ表現は自由である. FFT 乗算を行うためには, 精度はさることながら, ある程度大きな値を表現することができるデータ表現を用いる必要がある. このことを考慮し, 今回は浮動小数点表現を用いる. 浮動小数点表現は, 符号, 指数部, 仮数部で表現される. したがって, 求める最適なデータ表現とは, 最適な指数部長と仮数部長を持つ浮動小数点表現である. 最適な指数部長は演算中にオーバフローを起こさない最小の長さで決定すれば良い.

一方, 仮数部長は誤差の解析に基づいて決定する必要がある.

はじめに,Henrici の解析的な誤差見積もりを用いてデータ長を求める. データ表現の精度を表すマシンイプシロン ϵ_M は, 浮動小数点表現における仮数部の長さに依存し, 仮数部の bit 長を f とすると,

$$\epsilon_M = \frac{1}{2^f} \tag{8}$$

と表すことができる. これを,FFT 乗算と誤差の関係式 (2) に適用すると, 仮数部長, 桁数, 基数の関係を得る.

$$\frac{1}{2^f} = \frac{1}{192m^2(2\log_2 m + 7)r^2} \tag{9}$$

式 (9) から, 計算桁数 m と基数 r が与えられた場合の仮数部長を求めることができる.

しかし, 第 3 章で述べたように, 式 (9) で求められる仮数部長は, 最悪ケースの誤差計算に基づくものである. したがって, 実用的には最適であるとは言えない. そこで次に, 平山の指摘に基づき仮数部と指数部の長さを自由に変更できる浮動小数点表現を定義し, 最適な仮数部長を測定する数値実験を行った. ただし, 仮数部の基数は 2 とし,IEEE754 で規定されているような非正規化数は扱わないものとした. 最適な仮数部長の測定は, 最大値どうしの乗算で最大の誤差を与えるという性質に着目し, 桁数 m で表現できる最大値どうしの乗算において正しい結果が得られることを確認するという方法で行った. ただし, 基数 $r = 16$ とした. 結果を図 2 に示す. 図は, 横軸が桁数の対数, 縦軸が仮数部の bit 長を示している. また, グラフ中の × 印は実験によって測定した桁数に対する仮数部長を表している. 破線は式 (9) をプロットしたものである. グラフより, 実験によって得られた仮数部長は,Henrici の評価式から求めたものと比較して, かなり短いことがわかる.

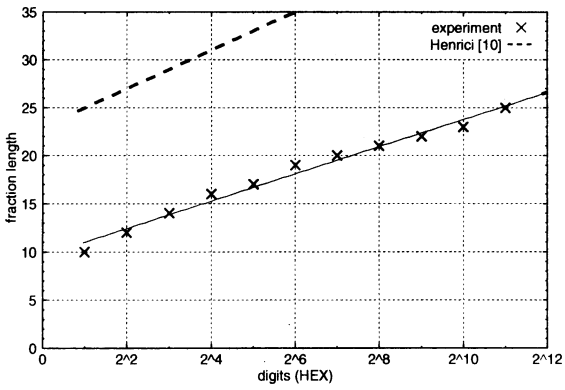

図 2 最小仮数部長と桁数の関係 ($r = 16$)

FFT 乗算をアプリケーションに利用することを考えた場合,2^{10} 桁以上の乗算を行うケースが多い. この実験から,2^{11} 桁どうしの乗算を行う場合, 仮数部長については,IEEE754 の 64bit 表現で規定されている 52bit より少ない 25bit でも正しい結果が得られることがわかる. 指数部長については,2^4 桁どうしの乗算までは 5bit を必要とする. また, 測定を行った 2^{11} までは 6bit で演算可能であった. したがって, 指数部も仮数部と同様に,IEEE754 64bit 表現で規定されている 11bit より短い 6bit で演算が可能である.

6. 実装結果と比較考察

前章の結果を踏まえ, 計算桁数に対する最適なデータ長を用いて FFT 乗算器を構成し, その面積と性能を調べた. 主な実装環境を表 1 に示す. 設計に用いたライブラリは, 日立製作所の仕様を元に VDEC(VLSI Design and Education Center) [13] が作成したものである.

表 1 実験環境

言語	Verilog-HDL
シミュレータ	Verilog-XL 04.10.001-p
論理合成系	Design Compiler 2003.6-SP1
ライブラリ	日立製作所 CMOS 0.18μm

面積に関する評価には, 図 1 の butterfly,inv-butterfly, complex multiplier,scaler,rounder-carrier の各モジュールを個別に合成した結果の合計を用いた. 速度に関する評価には, 面積と同様のモジュールの中で最大の遅延時間を用いた. 合成は遅延時間に対して最適になるような条件で行った. controller の面積と性能は, データ長に関わらずほぼ一定であり, その他のモジュールと比較して面積や遅延時間が非常に少ないため評価から除外した. また,memory モジュールについては, キャッシュと仮定しているため, 外部記憶の構造や計算桁数によって適切な構造が異なるので評価から除外した.

6.1 IEEE754 64bit 表現との比較

評価結果を図 3 に示す. 図の横軸は計算桁数の対数, 縦軸は面積を表している. 図 2 より,2^{11} 桁どうしの乗算に最適なデータ長は, 符号 bit, 指数部 6bit, 仮数部 25bit からなる 32bit の浮動小数点表現である. このデータ表現を用いて FFT 乗算器を構成すると, 面積が 5.2mm^2 であることがわかる.IEEE754 の 64bit 表現と同様に符号 bit, 指数部 11bit, 仮数部 52bit のデータ表現を用いて同様の実験を行った結果は文献 [14] で報告され

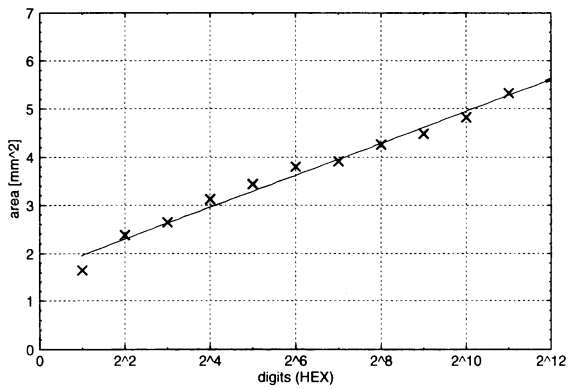

図 3 最適な FFT 乗算器の面積と桁数の関係 ($r = 16$)

ているが, 今回, 本実験と同じ条件の下で合成を行った結果, 面積は 15.3mm^2 になった. すなわち,2^{11} 桁どうしの乗算を行う FFT 乗算器の面積を IEEE754 の 64bit 表現と比較して約 1/3 に削減することができる.

次に, 桁数と速度について評価を行った. 結果を図 4 に示す. 図の横軸は桁数の対数であり, 縦軸は遅延時間を表している. 図

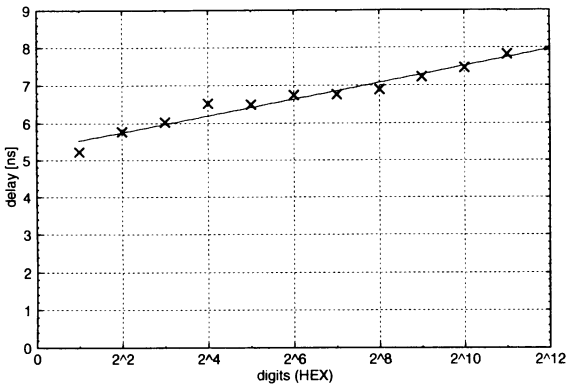

図 4 最適な FFT 乗算器の速度と桁数の関係 ($r = 16$)

から,2^{11} 桁どうしの乗算をするときの最大遅延時間は 7.8ns であることがわかる. 面積の評価と同様に IEEE754 の 64bit 表現を用いた場合の最大遅延時間を測定したところ,10.3ns であった. すなわち,2^{11} 桁どうしの乗算を行う場合,IEEE754 の 64bit 表現と比較して, 約 1.3 倍の性能を得ることができる.

6.2 既存の FFT ハードウェアとの比較

既存の FFT ハードウェアとの関連について述べる.FFT 乗算の実装において, 既存のハードウェアをそのまま利用したのでは正しい結果を得る保証がない. 正しい結果を得るための設計上の制約はコストの増加と性能の低下を招くと考えるのが妥当である。ここでは, そのことを調べるために, Miyamoto の FFT 実装 [7] と Bass の FFT 実装 [8] を例として面積と速度について本実装との比較を行った. 結果を表 2 にまとめる.

表 2 既存の FFT 実装との比較

	FFT		FFT mul.(2^8 桁)
	Miyamoto [7]	Bass [8]	Proposed
Rule[μm]	0.35	0.7/0.6	0.18
data length[bit]	36(fixed)	20(fixed)	28(floating)
Area[mm^2]	7.84	25	-
Area(mul.)[mm^2]	15.68(=7.84×2)	50(=25×2)	4.26
Clk[MHz]	133	173	138

FFT 乗算は,FFT の順変換と逆変換を行う必要があるので, 単純に考えると,FFT の 2 倍のコストが必要であるので, 表には面積を 2 倍にしたものも記載した. テクノロジーの違いがあり単純な比較はできないが, 表 2 より, 既存のハードウェア FFT をそのまま用いた場合と比較しても, 面積, 性能共に遜色ない実装が可能であると言える.

6.3 ソフトウェアとの比較

FFT 乗算のソフトウェア実装と速度比較を行った. 主な条件は表 3 のとおりである.

表 3 ソフトウェア実装の実行条件

CPU	Intel 社 Celeron 800MHz
OS	FreeBSD 4.8-RELEASE
コンパイラ	gcc 2.95.4
FFT ライブラリ	fftw Ver.2.1.5_1

FFT のライブラリには, 高性能な FFTW を用いた. また, CPU には Intel 社 [15] の CPU の中で, 本ハードウェア実装と同世代の 0.18μm プロセスを使用しているものを用いた. ハードウェアの実行時間は, 実行に必要なクロック数と最大遅延時間との積で求めた. また, ソフトウェアの実行時間は, 時間の測定誤差や同一システム上で動作している他プロセスの影響を最小におさえるために乗算 1000 回の実行時間を 10 回測定し, それらの平均をとった.

2^{11} 桁どうしの乗算に関して測定を行った結果, ハードウェアの速度が 0.74ms であったのに対して, ソフトウェアの速度は 7.8ms であった. この結果から, ソフトウェアと比較して, 約 11 倍の性能であることがわかった.

7. おわりに

本論文では,FFT 乗算における最適なデータ表現を誤差解析により求め, それを用いて,FFT 乗算器の最適化実装を行った.FFT 乗算器の FFT には, 最も基本的な Cooley-Tukey 型を用いた.

最適なデータ表現を求めるために, 最大値どうしの乗算が最大の誤差を与えるという点に着目し,m 桁における最大値どうしの乗算で正しい結果が得られるような仮数部長と指数部長を実験的に求めた. 実験結果から, 最適なデータ長を用いて FFT 乗算器のハードウェア実装を行った結果,IEEE754 の 64bit 表現を用いたものと比較して,2^{11} 桁どうしの乗算を行う FFT 乗算器の面積を約 1/3 に削減し, さらに性能を 1.3 倍向上させることができた. また, 設計上の制約があるにも関わらず, 本実装の面積速度は既存のハードウェア FFT と比較して, 遜色ないことを示した. ソフトウェアとの比較においては, 約 11 倍の速度向上が得られた.

今後は, さらなる高速化のために,FFT 乗算器に用いられている FFT アルゴリズムの改良や, 乗算器アーキテクチャの検討を行う必要がある. 特に,FFT アルゴリズムについてはより誤差の拡大が少ないようなものを用いることでデータ表現の bit 幅をさらに削減することができると考える. また,FFT 乗算を実際に応用し, その有効性を検討する.

謝　辞

本研究を進めるにあたり, 有益な議論をしていただいた電気通信大学の山本野人助教授と楯岡孝道博士, 東京大学の葛毅博士, 東京工科大学の月江伸弘先生, 電気通信大学阿部研究室の皆様に深く感謝する.

本研究におけるハードウェア設計は,VDEC を通じ, シノプシス株式会社とケイデンス株式会社の協力で行われたものである.

本研究は, 一部日本学術振興会科学研究費補助金 (基盤研究 (C)(2)16500026) によるものである. ここに記して謝意を表す.

文　献

[1] M. Agrawal, N. Kayal, and N. Saxena, "*PRIMES Is in P*," http://www.cse.iitk.ac.in/, 2002.

[2] 夛田光輝, 上島直樹, 茶谷芳裕, 鎌田弘之, "カオスシステムにおける有限桁演算の影響度の評価について," 第 17回 回路とシステム軽井沢ワークショップ論文集, pp.13-18, 2004.

[3] 月江伸弘, 小澤智, "カオス方程式の数値解における有効桁数の減少," 情報処理学会論文誌, Vol.44, No.11, pp.2778-2786, 2003.

[4] Dan Zuras, "More on Squaring and Multiplying Large Integer," *IEEE Trans. Computers*, Vol.43, No 8, pp.899-908, Aug. 1994.

[5] Matteo Frigo, "A Fast Fourier Transform Compiler," *ACM SIGPLAN Notices*, Vol. 39, Issue 4, pp.642-655, Apr. 2004.

[6] Joris van der Hoeven, "The Truncated Fourier Transform and Applications," *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, pp.290-296, 2004.

[7] N. Miyamoto, L. Karnan, K Maruo, K. Kotani and T. Ohmi, "A Small-Area High-Performance 512-Point 2-Dimensional FFT Single-Chip Processor," *Proceedings of the 2004 Conference on Asia South Pacific Design Automation : Electronic Design and Solution Fair 2004*, pp.537-538, Jan. 2004.

[8] Bevan M. Baas, "A Low-Power, High-Performance 1024-Point FFT Processor," *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 3, pp.380-387, Mar. 1999.

[9] D. E. Knuth, "*The Art of Computer Programming Volume 2 2nd Edition : Seminumerical Algorithms*," Addison-Wesley, 1981.

[10] Henrici P., "*Applied and Computational Complex Analysis*," Vol. 3, Chap. 13, John Wiley & Sons, 1986.

[11] 平山 弘, "FFT による高精度数の乗算," 情報処理学会 研究報告 *High Performance Computing*, Vol. 65, No. 65-6, pp. 27-32, 1997.

[12] J. W. Cooley and J. W. Tukey, "An Algorithm for Machine Computation of the Complex Fourier Series," *Mathematics of Computation*, Vol.19, pp.297-301, Apr. 1965.

[13] VLSI Design and Education Center Homepage, http://www.vdec.u-tokyo.ac.jp.

[14] 矢崎俊志, 阿部公輝, "高速 Fourier 変換を用いた多倍長乗算器の設計と評価および VLSI への実装," 電子情報通信学会技術報告, Vol.103, No.476, pp.253-258, Nov. 2003.

[15] Intel Corporation, http://www.intel.com.