C-003

# FFT を用いた畳み込み演算の FPGA 上でのハードウェア実装の検討
# Hardware implementation of convolutional operations using FFT on FPGA

滝田 涼介[1)]　　増田 信之[1)]
Ryosuke Takita　Nobuyuki Masuda

## 1　はじめに

近年，画像処理や機械学習の分野など，畳み込み演算は広範な用途で利用されている．しかし，この畳み込み演算は膨大な計算量があり，多くの時間及び，電力を消費する．そこで本研究ではこの畳み込み演算を高速フーリエ変換 (FFT) を用いて，FPGA (Field Programmable Gate Array) 上に専用ハードウェアの開発をすることで，計算量を削減し省電力で処理することを図った．

具体的には入力データの画素数が (32, 32) の FFT を行った後に畳み込み演算を行い，その後に逆 FFT (IFFT) を行った．

FPGA での FFT はデータの依存性から，データを 1 画素ずつ計算回路に供給し，内部でキャッシングし処理するのが一般的である．しかし，このような仕様は専用計算機での効率的な処理を目指す上で解決すべきである．そこで本研究では Vector Radix 法 [1] を用いることで並列化を行い，高速化を図った．用いた FPGA ボードは AMD Xilinx 社の Alveo U250 アクセラレータカードで，Verilog-HDL を用いてハードウェアの実装した．

## 2　2 次元の畳み込み演算

入力データ $x(i,j)$ とカーネル $w(m,n)$，出力データ $y(i,j)$ とすると 2 次元の畳み込みは以下のように示せる．

$$y(i,j) = (x * w)(i,j) = \sum_m \sum_n w(m,n) \bullet x(i-m, j-n)$$

次に上式を離散フーリエ変換を用いて表す．$x, w$ のフーリエ変換を $X, W$ とする．そうすると 2 次元の畳み込み演算はフーリエ変換したものどうしの積に等しくなる．式で表すと以下のようになる．このとき $\odot$ は要素積を表す．

$$(x * w)(i,j) = \mathcal{F}^{-1}[X \odot W] = \mathcal{F}^{-1}[\mathcal{F}[x] \odot \mathcal{F}[w]]$$

このとき，離散フーリエ変換で等しくするために $x, w$ の要素数をそれぞれ，$N_x, N_w$ とするとフーリエ変換時の要素数 $N_f$ は $N_f \geq N_x + N_w - 1$ でなければならない．通常の 2 次元の畳み込み演算の計算量が $\mathcal{O}(N_x^2 N_w^2)$ であるところ，高速フーリエ変換を用いることで計算量が $\mathcal{O}((N_f)^2 \log((N_f)^2))$ となり，計算時間が減少する．

## 3　実装

### 3.1　実装環境

本研究では入出力データの画素数を (32, 32), 符合付きの 16bit，固定小数点 8bit で専用計算機に送信した．データの送信は 1 クロックで 64bit 送信する．つまり 1 クロックで 4 画素のデータを送信する．FFT，畳み込み演算の際に整数部分を 15bit に拡張し，24bit で演算を行い，出力データは 16bit とした．また FFT，畳み込み演算の際は実部，虚部があるので 1 画素のデータは 48bit となる．

回路の記述は SystemVerilog 及び，Verilog-HDL を使い，AMD Xilinx 社の提供する開発環境ソフトウェア Vivado2020.1 を用った．本専用計算機の実装には Alveo U250 アクセラレータカードを使用した．このボードは UltraScale+ XCU250-2LFIGD2104E という FPGA が搭載されている．

### 3.2　全体の回路設計

本研究の回路設計を図 3.2 に示し，専用計算機の演算の流れについて述べる．まず，入力データをホスト PC から FPGA ボードの Block RAM に書き込む．次に書き込まれたデータを読み込み，2 次元 FFT を計算する．この演算には Vector-Radix 法を使用した．FFT 演算の後に各要素の積を計算し，IFFT を行う．最後に Block RAM に演算後のデータを書き込み，ホスト PC にデータを送信する．これが専用計算機の演算の流れである．ホスト PC と FPGA ボードの間の入出力データの転送は PCIe (PCI-Express) を使用し，データの転送部には AMD Xilinx 社の提供している DMA/Bridge Subsystem for PCI Express [2] を用いた．これを介して計算部の制御も行っている．カーネルの値は事前にホスト PC から送信し，データを読み込む設計とした．

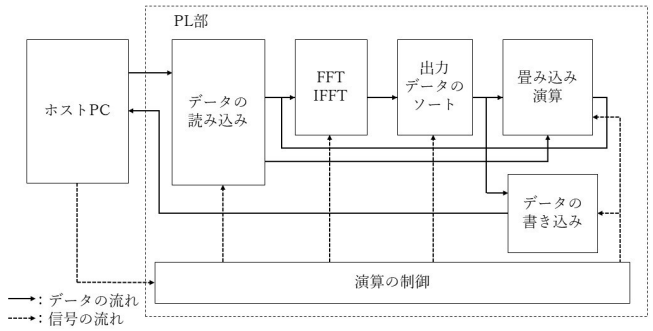


図 1　全体の回路

### 3.3　Vector-Radix 法

Vector Radix 法の FFT とは 1 次元の FFT を多次元に拡張したものである．行列を 1 次元ずつ行う FFT である Row Column 法よりも複素乗算が減少する．図 2 に基数 2 のバタフライ構造体を示す．図の通り $F^{00}_{N/2N/2}$ の入力において回転因子 $W_N$ と積をしないため，乗算を減少させ，リソースの削減が見込める．図 3 に回路図を示す．この手法を並列化し FPGA に実装した．(32, 32) の画素数であるので基数 2 のバタフライを 256 個並列化する．　この演算を $\log N$ 回繰り返すことで FFT が完了する．本設計では $N = 32$ であるので 5 回繰り返すことで FFT の演算が完了する．演算する前にデータを入れ

1)　東京理科大学

替えを行う．一度のバタフライの演算が 4 クロックであるので 20 クロックで FFT の計算を行える． また回転因子は以下のように定義される．

$$W_N^k = \exp^{-2\pi i \frac{k}{N}} = \cos\left(-2\pi\frac{k}{N}\right) + i\sin\left(-2\pi\frac{k}{N}\right)$$

上式より回転因子で使用する正弦，余弦の値は $N$ 個のみとなるので，事前に用意しテーブルから参照することで回転因子の値を得る． 回転因子は符合付き固定小数点 8bit の 10bit であり，入力データと積を取るときは実部，虚部それぞれを 34bit に一時的に拡張し， 演算後は 24bit に収縮する．

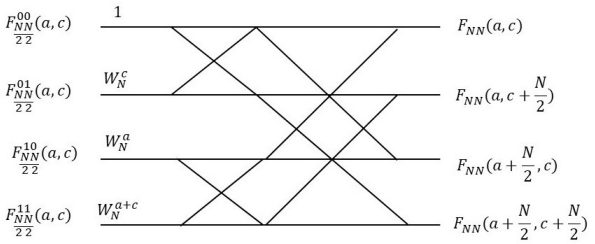


図 2　基数 2 の Vector Radix のバタフライ

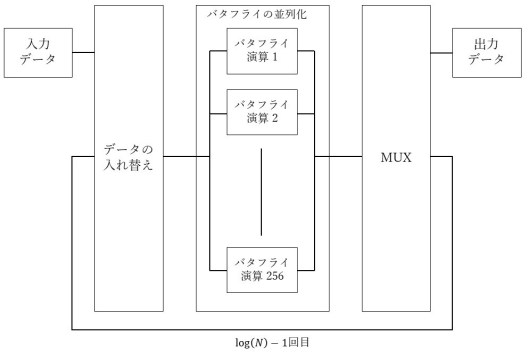


図 3　FPGA 上の FFT の回路図

### 3.4　2 次元の畳み込み部

2 次元畳み込み演算部の FPGA の設計について示す．フーリエ変換後の畳み込みは各要素の積となる．そのため，FPGA への実装は並列化できる． 本設計での要素積は (32, 32) の要素に対して 1 クロックで積をする設計とした． カーネルの FFT 後の値は事前に計算し，その値を Block RAM から読み込む．図 4 に要素積のイメージを示す．FFT 後の値は複素数であるので畳み込み演算部の要素積は複素数の積となる． したがって，FPGA に実装する際には 1 つの畳み込み演算に対して 4 つの積が行われる． FFT 部と同様に積をするときは 40bit に拡張され，演算後に 24bit に収縮する．

## 4　評価と結果

### 4.1　Vivado 上での実装

表 1 に本計算機の RTL 実装部のみのリソースを示す．実際のリソースは転送用の IP を含めるのでこれよりも増す． 更に FFT 部と畳み込み部のリソースも示す．1 つのバタフライに DSP が 12 個使われており，1 要素

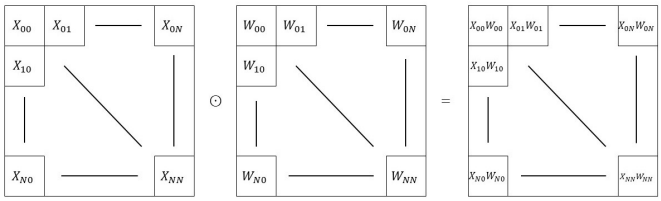


図 4　FFT 後の畳み込み演算

の畳み込み演算に 4 個の使われている． 動作周波数は Implementation 後に Vivado の Timing Report を確認したところ，15MHz であれば正常に動作することが分かった．

表 1　RTL 実装部のリソース使用量

| リソース名 | 使用量 | 使用率 [ % ] |
|---|---|---|
| LUT | 378,260 | 21.89 |
| FF | 145,051 | 4.20 |
| DSP | 7,168 | 58.33 |

表 2　FFT 部と畳み込み部の DSP の使用量

| 演算部 | 使用量 |
|---|---|
| FFT | 3,072 |
| 畳み込み | 4,096 |

## 5　まとめと今後の課題

本研究では入力データの画素数が (32, 32) に対して 2 次元の畳み込み演算を FFT を用いて実装した．
その一方で (32, 32) の画素数は非常に少ないため，畳み込みの定理に基づいて，カーネルの要素数に応じて元の入力データの画素数を減らす必要がある． 今後の課題として FPGA ボード上の DRAM, Ultra RAM 及び，Block RAM を使用して画素数を拡張することが挙げられる． 現状の設計では，RAM は読み込み，書き込み時の Block RAM の使用のみである．

画素数の拡張をするには具体的に，AMD Xilinx 社が提供する IP である MIG(Memory Interface Generator) を使用することで，Block RAM に収まらないデータは DRAM に格納する． これにより，より大きな画素数の演算が行える． また，要素数を拡張するにあたって，本研究の計算機では (32, 32) の画素数のみでデータののの入れ替えを行っている． そのため，FFT を行う際にデータの入れ替えが一定になるような入力データの送信を行うか，FPGA 側で RAM を使用することで値の読み込みを操作できる設計に変更する． 最終的には，通常の 2 次元の畳み込み演算と比較してリソースの使用量や動作速度を比較していきたい．

### 参考文献

[1] D. Harris, J. McClellan, D. Chan and H. Schuessler, ”Vector radix fast Fourier transform,” ICASSP ’77. IEEE International Conference on Acoustics, Speech, and Signal Processing, Hartford, CT, USA, 1977, pp. 548-551, doi: 10.1109/ICASSP.1977.1170349.

[2] AMD Xilinx, ”DMA for PCI Express (PCIe) Subsystem”, <https://japan.xilinx.com/products/intellectual-property/pcie-dma.html>, 2023 年 6 月 15 日閲覧