C-007

エッジ AI における畳み込み演算処理の高速化の検討
Consideration of Speeding Up Convolutional Calculation Processing in Edge AI

川﨑 健太† 中西 知嘉子‡
Kenta Kawasaki Chikako Nakanishi

1. はじめに

近年，エッジデバイスによる画像処理 AI の高速化に注目が集まっている．しかし，エッジデバイスにAI処理をさせるにはリソースや電力等の制約があり，高速動作をさせることが難しい．高速化に関する研究には，ネットワークの改良や量子化によるデータ数削減[1]や全処理の回路化による高速化[2]などがある．しかし，どの手法においても開発コストがかかることや精度劣化，柔軟性など様々な問題点がある．

そこで，我々は CPU と FPGA が同一チップ上にある SoC FPGA を用いて，畳み込み層の処理を行う回路を FPGA 上に作成し，ハードウェアがソフトウェアと連携して畳み込み処理をアクセラレートするソフト・ハード協調設計技術の研究に取り組んでいる．これらの研究において，開発期間や開発コストを抑えるため，ネットワークの構造をそのまま用いることで精度の保ちながら，SoC FPGA を用いることでAIの急速な発展に対応することが出来た．その結果，ある程度の高速化が実現できている [3][4][5]．しかし， 畳み込み処理の演算部において，浮動小数点を使用しているため，演算の高速化には限度がある．そのため，固定小数点を用いて演算処理を行うことを検討することにした．固定小数点による演算は，桁落ち，桁溢れによる精度劣化に繋がり，最終の推論精度が低下するという問題が発生する．

そのため，本研究では，速度向上に直結する FPGA 回路内の積和演算部のみを固定小数点を用いて，その他の処理は全て浮動小数点のまま扱うことで，速度と精度の両立を目指す．

2. 分析

2.1 使用ネットワークモデル

今回評価に使用するネットワークモデルは，Oxford 大学の研究グループが 2014 年の ILSVR で提案し好成績を収めた物体認識モデルである VGG16[6]，2018 年に発表された YOLOv3 をニューラルネットワークライブラリである keras で動作可能な物体検出モデル keras-yolo3-tiny[7](以降，yolov3-tiny)，2015 年に発表された残差構造を持つ物体認識モデルである ResNet50[8]を用いて検証・評価を行った．各モデルの学習に使用されているデータセットは順に Cifar10, MS Coco，ImageNet である．

我々は，ソフト・ハードの協調動作を行うため，C++言語で推論処理を行うことができる Ceras[9]を開発している．回路設計は高位合成を用いている．C++言語での推論処理は，高位合成前にソフトウェアとハードウェアのインターフェースを検証できるという利点がある．そのため，回路実装後のソフトウェアとハードウェアの協調動作が保証でき，開発効率を高めることができる．

2.2 速度計測

今回，固定小数点化を行う回路については，文献[5][10]の 32bit 浮動小数点回路を使用する．回路情報を表 2.1 に示す．この回路において，回路演算にかかる時間を調査した結果を表 2.2 に示す.

表 2.2 から，VGG16 の回路内演算処理は 300ms，yolov3-tiny では約 1480ms，ResNet50 では約 2300ms も時間かかっている．この時間を，演算処理を固定小数点化にすることで高速化を図る．

表 2.1 文献[5][10]の浮動小数点畳み込み演算回路の仕様

フィルタサイズ	3×3，1×1
ストライド	1，2
入力データ	66×66×128
フィルタサイズ	3×3×64×128 1×1×64×1152
バイアスサイズ	64
積和演算の並列度	3×3 の場合：4 1×1 の場合：36

表 2.2 各ネットワークにおける畳み込み演算回路の処理時間[ms]

使用 ネットワーク名	浮動小数回路 畳み込み演算処理時間
VGG16	288.855
yolov3-tiny	1480.034
ResNet50	2288.302

2.3 精度検証

精度評価においては，平均絶対パーセント誤差(MAPE)を用いて評価を行った．計算式を以下の式(1)に示す．式内の n はデータ数，A_t は浮動小数点による演算結果，F_t は固定小数点による演算結果を示している．

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|A_t - F_t|}{A_t} \quad (1)$$

† 大阪工業大学 情報科学研究科 情報科学専攻
Graduate School of Information Science and Technology
Osaka Institute of Technology
‡ 大阪工業大学 情報科学部 情報知能学科
Department of Information and Computer Science
Osaka Institute of Technology

Copyright © 2022 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

精度評価についての指標として浮動小数点回路の推論結果と，32bit 幅の整数部 8bit 固定小数点回路，32bit 幅の整数部 15bit 固定小数点回路の MAPE で精度検証を行った．結果を表 2.3 に示す．固定小数点については，3.2 節で後述する．

表 2.3 から，浮動小数点と 2 通りの任意整数ビット幅の固定小数点とで，約 0.01 ~ 4%の間で誤差が 発生していることが確認できる．VGG16 においては，整数部 8bit での固定小数の場合，正常な出力結果が得られなかった．

そのため，誤差を最小限に抑えるための手法を考察し，固定小数点化を行うことにした．

表 2.3 浮動小数点と任意整数ビット固定小数点の精度評価の差(単位：%)

	VGG16	yolov3-tiny	ResNet50
32bit 固定小数点 整数部 8bit	精度評価不可	0.01706	0.02281
32bit 固定小数点 整数部 15bit	3.95029	2.11736	3.21512

3. 回路設計

3.1 畳み込み演算

畳み込み演算について，図 3.1，図 3.2 を交えながら説明を行う．ここで，入力データサイズを 4×4×3，フィルタサイズを 3×3×3，ストライド 1 として説明を行う．

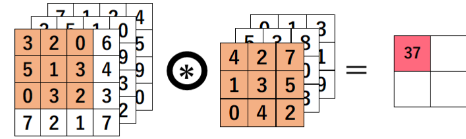
畳み込み演算の処理手順を入力データの左上から以下の I ~ III の流れで行う．

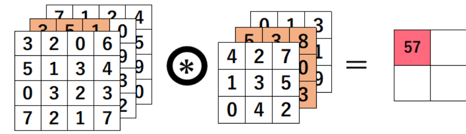
I. 入力データにおいてフィルタと同じサイズ分の要素を取り出し，フィルタと同じ箇所の要素同士の掛け算を行う．

II. I で得られた演算結果をチャネル方向に対しても同様に行う．そして得られた結果は，1 つ前の演算結果に加算する．

III. 入力データの領域を横(もしくは縦)方向に 1 つ移動し，I，II と同様の演算を行う．この演算を入力データの右下まで繰り返し行う．

上記手順より，1 つのデータの積和回数は，フィルタサイズとフィルタ枚数の和となる．これら全てを入力データに対して行う必要がある．膨大な回数の積和演算を行う畳み込み処理において，1 回あたりの積和演算処理を高速化することが出来れば，全体として大きな速度向上を得ることが可能になる．





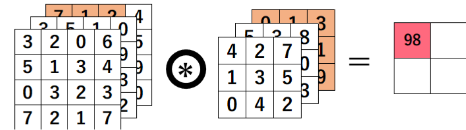
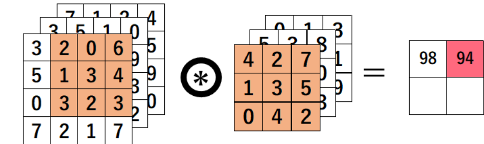


図 3.1 畳み込み演算処理(手順 I, II)



繰り返し

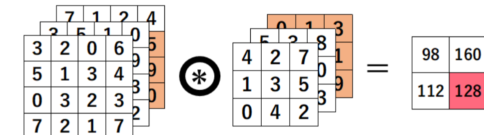


図 3.2 畳み込み演算処理(手順 III)

3.2 固定小数点

プログラムの小数点表現には 2 種類存在しており，浮動小数点と固定小数点がある．各型のデータ構造を図 3.3, 図 3.4 に示す．浮動小数点型は，指数部の値を元に仮数部のデータの桁合わせ処理を行い小数点の位置を決定する．これにより，幅広いデータ表現を行うことが可能である．しかし，桁合わせ処理を演算ごとに必要とするため，処理速度が低下する．

固定小数点型は，浮動小数点と異なり予め小数点の位置を固定する方式である．そのため桁合わせ処理の速度低下がない．しかし，小数点の位置が固定されていることで，表現できる値の範囲が狭く，桁溢れ・情報落ちが発生してしまう．

図 3.3 32bit 浮動小数点型のデータ構造

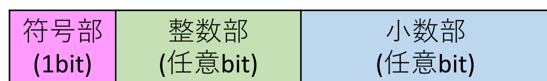


図 3.4 32bit 固定小数点型のデータ構造

Copyright © 2022 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

3.3 桁溢れ・桁落ち防止

32bit 固定小数点の積和演算，特に掛け算においては２つのデータの積を求めた後の bit 列は切り捨て，もしくは丸め上げを行うためデータの桁落ちが生じてしまう．固定小数点型の情報落ちを防止するため，積和演算に 32bit から 48bit にデータ幅を拡張する．また，演算後の結果保持についても 48bit のままとする．浮動小数点型の仮数部が 23bit であるため，単純に倍とするだけで必要なデータの情報落ちが発生しにくくなる．また，48bit という数値は PFGA 内に DSP48 という演算処理機構が存在するため，48bit での演算が負荷を増加させずに実行できると考えた．10bit 固定小数点と 15bit にデータ拡張した固定小数点の例を図 3.5 に示す．

拡張した bit を整数部および小数部にどのように割り当てを行うか検討するため，データの最大値の調査を行った．そして，最大値から整数部に割り当てるビット数を決定する．回路設計に用いた C++言語のプログラムを用いて，入力データ，フィルタ，積和演算後，回路からの転送前の 4 箇所を調査した．その結果を表 3.1 に示す．表 3.1 の調査結果から，ネットワークごとに整数部の変更をする必要があることがわかった．そこで，ネットワークごとに回路設計を行った．

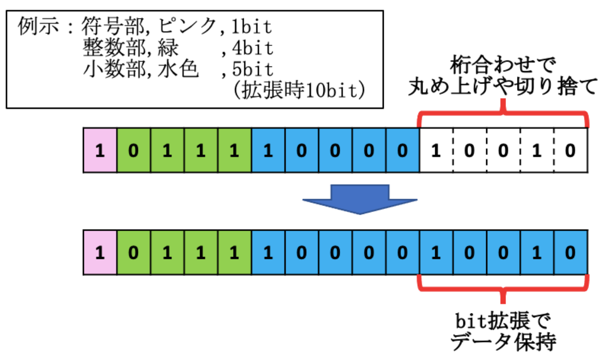


図 3.5 bit 拡張によるデータ保持

表 3.1 各ネットワークにおいて最大値を計測した箇所と整数部に必要な bit 幅

使用 ネットワーク名	最大値 計測箇所	整数部 必要 bit 幅
VGG16	積和演算後	13bit
yolov3-tiny	積和演算後	14bit
ResNet50	入力データ	8bit

3.4 回路内でのデータ保持

固定小数点を FPGA 回路に組み込んだ場合，浮動小数点の桁合わせ処理が存在しないため，演算機構 DSP48 のリソースの削減に繋がり，並列度を増加できる．並列度向上によるメモリアクセスの集中回避のため，一度にアクセスするフィルタの各データを 1 パックとしてまとめ，BRAM に 1 ラインで保持できるようにする．1 パック化により，複数のデータアクセスが無くなり，必要なデータが 1 度に得られるようになる．図 3.6 に 9 つのデータの場合の概念図を示す．

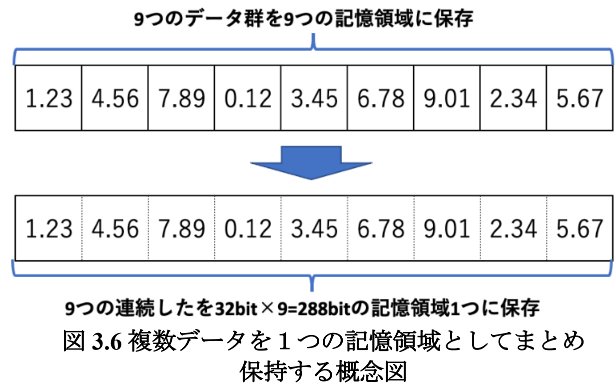


図 3.6 複数データを 1 つの記憶領域としてまとめ保持する概念図

3.5 データの分割処理による転送

1 つの層に用いるフィルタ，バイアス，入力データは膨大であり，すべてのデータを演算回路に保持させることは，回路内 BRAM のリソース量に上限が存在するため困難である．本研究で扱う回路においてもリソース量の上限を超えないよう保持可能サイズを設定している．そのため，ソフトウェア側でデータ分割を行い共有メモリへ転送，また回路処理終了後，演算結果のデータをソフトウェア部において合成処理を行う必要もある．入力データの分割処理についてフィルタサイズ 1×1 で行った場合の処理を図 3.7 に示す．

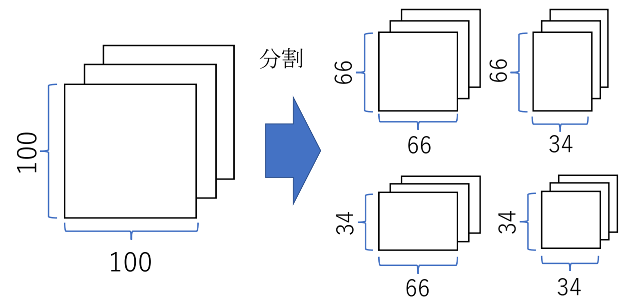


図 3.7 データ分割処理

4. 回路作成

4.1 型変換処理について

ソフトウェア側では浮動小数点のみを扱いたいため，回路内のみで固定小数点処理は完結させる設計を行う．そのため，型変換処理は全て FPGA 回路内で行う．入力されたデータに関しては全て 32bit 固定小数点に変換をする．変換処理については，表 4.2 に示す箇所で行う．回路内の BRAM に保持するバイアス，フィルタについては，回路内に転送されてきたときに変換処理を行う．入力データについては，9 つのデータが揃うたびに積和演算を行う回路構成にしているため，処理の簡単化のために積和演算の前に必要なデータ数分だけを固定小数点化する．

Copyright © 2022 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

浮動小数点で生成した場合と，固定小数点化をした場合との回路の各リソースの使用量を表 4.2 に示す．表 4.2 から使用リソース量が約 20%も削減できていることが確認できる．

表 4.1 固定小数点へ変換を行う箇所

データ	変換箇所
バイアス	回路へ転送された直後
フィルタ	回路へ転送された直後
入力データ	積和演算直前

表 4.2 型変換による使用リソース量の違い(単位：%)

	BRAM	DSP48	FF	LUT
浮動小数点回路	71	50	33	34
固定小数点回路	71	30	24	31

4.2 BRAM に保持するデータについて

BRAM に保持するデータの内，フィルタについてはデータの量が多いためメモリアクセスが集中してしまう．そのため，3.4 節で述べたデータの 1 パック化処理を行った．1 パックを行うにあたり，3×3 のフィルタ 1 枚に必要なデータ数が 9 つであるため，9 つのデータを 1 パック化した．データ保持の手法を変更した回路リソースの結果を表 4.3 に示す．変更を行った BRAM のリソース量を確認すると BRAM のリソース量が 43%まで削減した．

表 4.4 データ保持手法によるリソース量(単位：%)

	BRAM	DSP48	FF	LUT
変更前	71	30	24	31
変更後	43	30	15	32

4.3 演算並列向上

高速化には，積和演算の並列度を向上させることが有効である．しかし，並列度をあげるにはリソース量について考えて行う必要がある．4.1 節で DSP48 のリソース量を削減したことにより，並列度を向上させることが可能となった．そのため，積和演算の並列度を 3×3 の場合 4 から 8 に，1×1 の場合 36 から 72 に変更した．また，4.2 節の処理による BRAM のリソース量の削減から，回路内で保持できるフィルタのサイズを 64 から 120 に変更を行った．再変更後のリソース量を表 4.3 に示す．

表 4.4 並列度向上によるリソース量の違い(単位：%)

	BRAM	DSP48	FF	LUT
フィルタサイズ 積和演算並列度 変更前	43	30	15	32
フィルタサイズ 積和演算並列度 変更後	75	60	23	38

4.4 データ転送

本研究では，データの転送に Data Memory Access(以降，DMA)転送を用いる．ソフトウェアから共有メモリにデータを書き込み，データの転送については DMA 転送回路が行う．DMA 転送を行うには，共有メモリに書き込んだ連続データの先頭アドレスと長さ(データのバイト数)の情報を AXI DMA に知らせる必要がある． 図 4.1 に DMA 転送を実現する回路配線図を示す．データ転送には，AXI4 Stream を利用し，DMA レジスタへのアクセスには，AXI4 Lite を利用する．

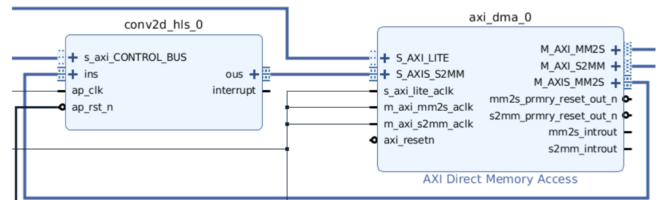


図 4.1 DMA と畳み込み回路を接続した配線図

4.5 キャッシュ管理

共有メモリを使用したデータ転送を行う場合，キャッシュとメモリの値一貫性の問題が発生する．CPU は，キャッシュを通じてメモリ上のデータをアクセスするが，回路は直接メモリにデータのアクセスを行う．そのため，データの一貫性を保つためのキャッシュ管理を行う必要がある．

ソフトウェア側でキャッシュ管理を行った場合，キャッシュとメモリの同期管理の必要があり，処理時間が長くなってしまう．そこで，回路側でキャッシュ管理を行うように変更をすることにより，キャッシュ管理にかかる処理時間の短縮を図る．

回路でキャッシュ管理を行うための回路配線図を図 4.2 に示す．Constant IP2 つを AXI Interconnect IP に接続し，片方の Constant IP を 4bit 幅の定数値 0xf，もう一つを 3bit 幅の定数値 0x2 に設定する [11].

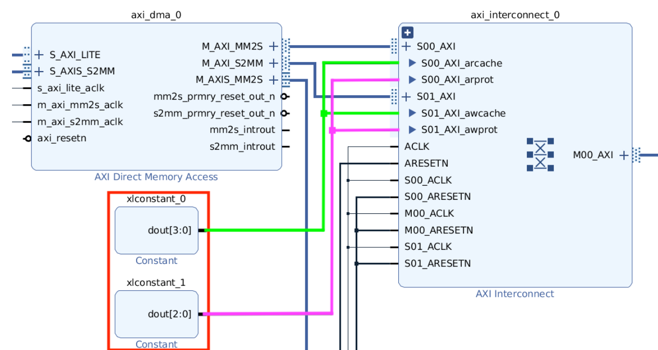


図 4.2 キャッシュ管理を実現する回路配線図

Copyright © 2022 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

5. 使用機材

5.1 開発環境

回路開発にあたっては，C++言語でアルゴリズムを設計した．設計後，Xilinx 社が提供している Vivado HLS 2019.2 を用いて高位合成を行い，HDL 記述言語化を行った．高位合成を用いることで，開発期間を短縮している．また，PL 部と PS 部の接続にあたっては同社が提供している Vivado 2019.2 を用いた．

5.2 評価使用機材

計測に使用した機材は，Xilinx 社の SoC PFGA という集積回路を搭載した Ultra96-V2[12]である． SoC FPGA の特徴として， FPGA と CPU の接続に広帯域バスを含めた内部バスが接続されおり，高速なデータ転送が可能であることがある．そのため，ソフトウェアと回路の協調動作が可能であるため検証に適していると考え使用する．Ultra96-V2 の性能を図 5.1 に示す．

表 5.1 Ultra 96-V2 の性能

<table>
<tr><td rowspan="2">CPU</td><td>Application Processing Unit</td><td colspan="4">Quad-core Arm Cortex-A53 MPCore with CoreSight; 32KB/32KB L1 Cache, 1MB L2 Cache</td></tr>
<tr><td>Real-Time Processing Unit</td><td colspan="4">Dual-core Arm Cortex-R5F with CoreSight; 32KB/32KB L1 Cache, and TCM</td></tr>
<tr><td colspan="2">Embedded and External Memory</td><td colspan="4">256KB On-Chip Memory w/ECC; External DDR4; DDR3; DDR3L; LPDDR4; LPDDR3; External Quad-SPI; NAND; eMMC</td></tr>
<tr><td colspan="2">RAM</td><td colspan="4">Micron 2GB LPDDR4 Memory</td></tr>
<tr><td colspan="2" rowspan="4">Programable Logic</td><td>System Logic Cells</td><td>154,350</td><td>Distributed RAM</td><td>1.8 Mb</td></tr>
<tr><td>CLB Flip-Flops</td><td>141,120</td><td>Block RAM Blocks</td><td>216</td></tr>
<tr><td>CLB LUTs</td><td>70,560</td><td>Block RAM</td><td>7.6 Mb</td></tr>
<tr><td>DSP Slices</td><td colspan="3">360</td></tr>
</table>

6. 評価と結果

6.1 精度誤差

作成した FPGA 回路を用いた各ネットワークの精度評価について，式(1)の平均絶対パーセント誤差を用いて計算した結果を表 6.1 に示す．表 6.1 から，各ネットワークの誤差が約 0.005%程度であることが分かる．

表 6.1 設計した固定小数点回路と浮動小数点回路の誤差

使用ネットワーク名	誤差(単位：%)
VGG16	0.00042
yolov3-tiny	0.00495
ResNet50	0.00641

6.2 処理速度

次に，回路の演算処理の処理速度について計測した結果とその倍率を表 6.2 に示す．表 6.2 より各ネットワークにおいて 2.1 倍以上の高速化を行うことが出来ていることがわかる．

表 6.2 処理時間(単位：ms)と倍率

使用ネットワーク名	浮動小数回路畳み込み演算速度	固定小数回路畳み込み演算速度	倍率
VGG16	288.855	128.587	2.246
yolov3-tiny	1480.034	591.272	2.503
ResNet50	2288.302	1054.27	2.171

7. 今後の課題

7.1 固定小数点回路の整数部可変化

本研究において，ネットワークごとに整数部のbit幅を決定し，回路設計を行った．そのため，使用するネットワークごとに回路の生成をしなければならないという問題があった．

汎用性のある回路の作成を行うため，ソフトウェアから回路の固定小数点の整数bit数を制御できるようにする方法が考えられる．実現するには，回路内部の固定小数点の型変換処理を可変にする必要性があり，回路設計に工夫が必要となる．

7.2 データ転送手法

今回は回路の高速化に焦点を当てたが，ソフトウェアから共有メモリへのデータ書き込み時間についても考えていく必要がある．表 7.1 に共有メモリへの書き込み時間を示す．書き込みについて各ネットワークの調査を行った結果，入力データの転送において同じデータを複数回再転送しなければならないことが判明した．図 7.1 にデータ分割のフローチャートを示す．現状のアルゴリズムでは，回路に転送するデータを，回路を起動するたびに共有メモリに書き込むような仕様となっている．フィルタを分割するたびに，内部保持をしない入力データの再送が必要になり，共有メモリに再書き込みする必要がある．共有メモリへの書き込みに時間が掛かってしまうと，回路処理の高速化の効果が薄れる．そのため，再書き込みを行わない手法の検討をする必要がある．

表 7.1 共有メモリへの書き込み時間[ms]

使用ネットワーク名	ソフト部から共有メモリへの書き込み
VGG16	212.722
yolov3-tiny	479.166
ResNet50	88183.276

Copyright © 2022 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

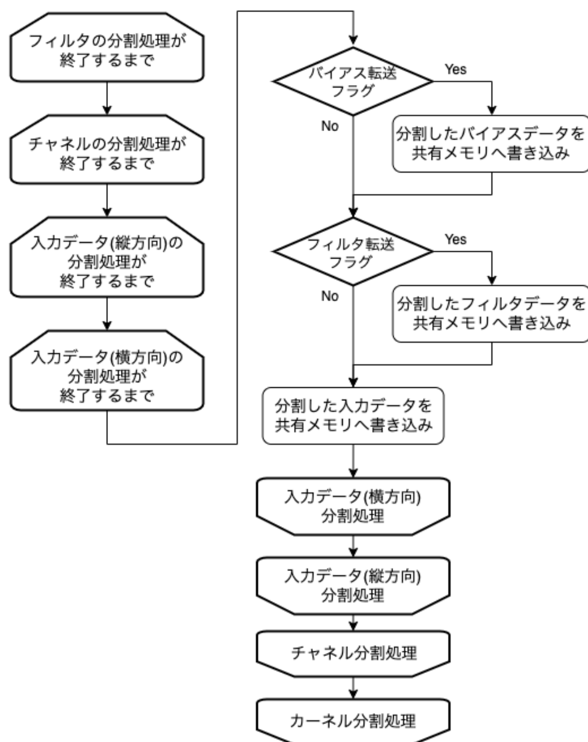


図 7.1 データ分割と共有メモリへ書き込むための アルゴリズム

8. 結論

本研究において，FPGA 回路の固定小数点化による回路の高速化，及びネットワークの精度保持を実現するための回路開発と検証，評価を行った．結果として，畳み込み演算の積和演算部のみを固定小数点で演算を行い，残りの全ての処理は浮動小数点を用いることにより精度劣化を約 0.005%程度に抑えることが出来た．また，3 種類のネットワークの畳み込み演算処理を行う回路において，浮動小数点回路の 2.1 倍以上の速度で高速化を実現することが出来た．

今後の展望として，回路の固定小数点整数部を可変化する処理を加えてネットワークごとに回路変更を不要にすることや，データを共有メモリへ再書き込みしない手法，他種の畳み込み処理に対しての動作・検証などが挙げられるため，これら問題の解消や検証を行っていく．

参考文献

[1] 近藤良太，平川翼，山下隆義，藤吉弘亘，Binary-decomposed DCNN におけるハイパーパラメータの自動最適化，精密工学会 (2019).

[2] Alireza Ghaffari, Yvon Savaria，”Toward Designing a General Framework for Implementation of Convolutional Neural Networks on FPGA”，arXiv：2004.04641v2 (2020).

[3] 岩本征弥，中西知嘉子，深層学習による推論処理の高速化手法の検討，一般社団法人電子情報通信学会 (2022).

[4] 川上智也，中西知嘉子，ハードウェアとソフトウェアの協調動作による AI 推論処理の高速化の検討，一般社団法人電子情報通信学会 (2022).

[5] 大戸彰馬，中西知嘉子，推論処理における畳み込み処理の回路化の検討，一般社団法人電子情報通信学会 (2022).

[6] Karen Simonyan and Andrew Zisserman，”Very Deep Convolutional Networks for Large-Scale Image Recognition”, arXiv：1409.1556 (2014)

[7] keras-yolo3, https://github.com/qqwweee/keras-yolo3 (2018).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun，”Deep Residual Learning for Image Recognition”， arXiv:1512.03385 (2015).

[9] 西岡駿, 中西知嘉子，機械学習ライブラリの C 言語化の実現，一般社団法人電子情報通信学会 (2021).

[10] 大戸彰馬，SoCFPGA によるディープラーニングのリアルタイムの実現手法の検討，大阪工業大学修士論文 (2022).

[11] 奥畑宏之，“最強 FPGA ボードで人工知能カリカリ画像認識”，Interface 第 2019 年 1 月，pp.62-90，2018 年 11 月．

[12] Avnet，AES-ULTRA96-V2-G, https://www.avnet.com/shop/japan/products/avnet-engineering-services/aes-ultra96-v2-g-3074457345644050668/ .

Copyright © 2022 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.