C-020

エッジ AI における YOLOX の比較評価
Comparative evaluation of YOLOX in Edge AI

吉田 裕紀‡　　中西 知嘉子‡
Hiroki Yoshida　Chikako Nakanishi

1. はじめに

近年, 生成 AI をはじめとした AI 技術が飛躍的な進化を続けている. 中でも小型のエッジデバイス上で AI 処理を行う「エッジ AI」の需要が高まっている. しかし, 一般的に小型デバイスは性能が低く, 複雑かつ膨大な演算が必要な AI 処理を高速に行うことは難しい.

そこで本研究では SoC FPGA を用い, AI 処理内の高負荷な箇所のみを FPGA 回路で行うことで AI 処理の高速化を行った. SoC FPGA は CPU と FPGA が 1 つのチップ上に統合された製品であり, 互いに高速なバスで接続されていることで, 協調動作が行いやすいという特徴を持っている. この特徴に着目し, AI 処理全体のうち, 大半の処理を CPU 上で動作させつつ, 高負荷な特定の箇所のみを回路化し FPGA 上で動作させることで, 物体検出 AI の高速化を図った[1].

また, 推論時間全体で CPU – FPGA 間でのデータ共有に要する時間の割合が高くなったことを受け, 高速化を進める中で発生した無駄なデータコピー処理の削減も行った[2].

本発表では, 研究手法を取り巻く環境について知見を広げることを目的に, 性質が異なるデバイスで AI 処理を動作させた結果を比較しまとめる.

2. 開発環境・使用モデル

2.1 開発環境

エッジデバイスに Avnet 社より販売されている Ultra96-V2[3]を用いた. Ultra96-V2 には, Raspberry Pi3 Model B+と同等の CPU が搭載されている. OS は Ultra96-V2 向けの Debian11 を使用した.

2.2 YOLOX

YOLOX[4]は物体検出 AI の YOLO シリーズより, 2021 年に発表されたモデルである. YOLO シリーズは, 他の物体検出モデルと比較し, 推論速度が高速なことが特徴として挙げられる. YOLOX には計算量に応じた複数のモデルが存在しており, 本研究では高精度向け Standard Model に位置付けられているものから, YOLOX-s モデルを使用した.

3. 実装手法

3.1 回路部

3.1.1 高位合成による回路生成

FPGA 上に実装する回路は, C++言語で記述したコードを高位合成によって RTL 言語へと変化することで作成した. 高位合成には Vitis HLS 2020.2[5], 回路設計には Vivado Design Suite 2020.2[6]を使用した. また Ultra96-V2 には Debian のデバイスツリー・オーバーレイを用い回路を実装している.

3.1.2 使用回路

使用回路は, Conv2D 層及び活性化関数処理を行うものである. 先行研究[7]によって開発された Conv2D 層向け汎用回路をベースに, YOLOX-s で使用することを想定し作成した. 実装されている活性化関数は Hard-Swish 関数, Hard-Sigmoid 関数, Linear 関数であり, 活性化関数の種類の選択や, 活性化関数処理を行うか否かを CPU による制御で決定できるよう設計されている. また, 回路作成に使用できるリソースの関係から, 一度に保持できるデータ量の保持上限値が設定されている. 表 1 に回路の仕様を示す.

表 1 回路の仕様

入力データサイズ上限	82
カーネル数上限	128
1×1チャネル数上限	576
3×3チャネル数上限	64

3.2 ソフトウェア部

3.2.1 データ分割処理

入力データの縦×横サイズや, チャネル数, カーネル数が表 1 の各上限を上回る場合, それぞれ CPU 上でデータの分割処理を行う. 分割されたデータは順に回路で演算され, 分割された全てのデータの演算が終了した後, CPU 上で演算結果の合成を行う. チャネル数が上限を上回る場合のみ, 活性化関数処理は CPU 上で行い, その他の分割では回路内で活性化関数処理を行う. 図 1 に縦×横サイズが設定上限を上回る場合の例を示す.

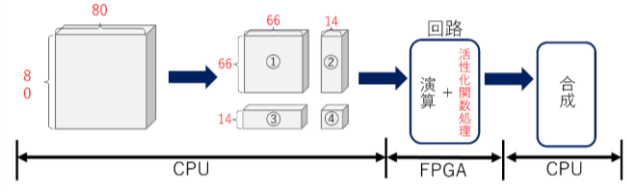

図 1 サイズ分割の例

ただし, データ共有に要する時間を削減するため, 回路の連続実行が可能な場合など, 特定の条件を満たす場合には CPU による演算結果の合成は行わない. その場合には次層でデータ読み出しと同時に整形を実施する.

3.2.2 Ceras

Ceras[8]は弊研究室にて開発された, 学習済み ONNX モデルの推論を C++言語で実行するライブラリである. C++言語の標準モジュールのみで構成されており, 環境構築が不要な点や, 層内部の処理単位にまで着目できる点, 開発環境を C++言語で統一できる点から使用した.

本研究では, Ultra96-V2 上で動作させるモデルに対し Ceras を用いている.

4. 比較

推論処理の開始から終了までに要した時間, 推論処理実行時の消費電力を比較する. また, AI モデルを動作させるうえで発生する作業コストも考慮し, 結果と併せて考察する.

4.1 比較回路・機材・モデル

4.1.1 比較に使用する機材

性質の異なるデバイスとして AMD ザイリンクス社より販売されている KV260[9]を用いた. KV260 は FPGA を搭載したカメラ AI 向け SOM ボードで, AI 処理に特化したプロセッサである DPU を用いた動作が可能となっている.

4.1.2 比較に使用するモデル

本研究の提案手法と KV260 上での AI 処理の比較には, Vitis AI[10]向けに量子化済みのモデルが公開されている YOLOX-Nano を用いた.

本研究手法で動作させるモデルは, 回路内を含む全ての演算を 32bit 浮動小数点型で行うのに対し, KV260 で動作させるものは 8bit 量子化が施されたモデルとなっている.

以降, YOLOX-s を-s, YOLOX-Nano を-Nano と略す.

4.2 比較項目

比較項目は以下である.

① 本研究手法による-s, -Nano の動作比較
② DPU を用いた場合との動作比較 (-Nano)

5. 結果・考察

5.1 -s, -Nano の比較

本研究手法を用い, 両モデルを動作させた結果を表 2 に示す. -Nano は, 推論結果が大きくずれる問題が発生したため, 回路内に実装されている Hard-Swish 関数, Hard-Sigmoid 関数は使用していない. かわりに Swish 関数, Sigmoid 関数による処理を CPU 上で行った.

表 2 両モデルの動作結果

	-s	-Nano
推論時間(ms)	7,481	2,332
消費電力(W)	7.0	6.5

表 2 より, -Nano の推論時間は 2,332ms であり-s の 3 分の 1 以下となっていた. また, 消費電力にも 0.5W の差があることがわかった.

次に推論時間の内訳として, 回路実行時間と CPU 処理時間を調べた結果を表 3 に示す.

表 3 推論時間の内訳

	-s	-Nano
回路実行 (ms)	3,593(48%)	299(12%)
CPU処理 (ms)	3,888(52%)	2033(88%)
計 (ms)	7,481	2,332

表 3 を確認すると, -s では回路実行時間が約半数の 48%を占めていることに対し, -Nano は 12%と割合が低くなっていることがわかる. -Nano の回路実行時間が少ない要因として, 使用回路が GroupConv2D 層に対応していないこと, 活性化関数処理を回路外で行ったことが考えられる. またモデルによる差として Conv2D 層自体の負荷の差が考えられる. 回路を用いて処理が行われた層数は両モデル共に 83 層と同じであった. しかし Conv2D 層の平均チャネル数が, 約 202 チャネル(-s), 約 104 チャネル(-Nano)と大きく異なっており, 処理内容の負荷にも大きな差があったと言える.

5.2 DPU を用いた場合との比較

-Nano を本研究手法で動作させた場合と KV260 上で DPU を用いて動作させた場合の結果は表 4 の通りとなった.

表 4 -Nano を用いた KV260 との比較

	本研究手法 (Ultra96-V2)	KV260
推論時間(ms)	2,332	503
消費電力(W)	6.5	10.3

推論時間のみで比較すると, KV260 は本研究手法に対し 4 分の 1 以下であった. しかし, KV260 の動作消費電力は 10W を超え, 本研究手法の 1.5 倍以上であった. また KV260 上にモデルを実装する過程にもツールの複雑さによる実装難易度が問題として存在する. 特に公式より提供されていない AI モデルを, Vitis AI を用い KV260 上で動作させるには, 量子化, コンパイルで用いるファイルの作成が必要となり, 少なくない作業コストが発生する.

6. まとめ

本発表では, 知見を広げる目的で本研究手法に対し 2 種類の比較を行った. 引き続き比較評価を行い, 今後は性質の異なるデバイスのみでなく, 本研究の回路とは異なるシステム設計で開発された回路との比較も行う予定である.

参考文献

[1] 吉田裕紀, 中西知嘉子, “物体検出モデル「YOLOX」のエッジデバイス上での動作高速化手法の検討”, 信学技報, Vol. 123, No. 71, RECONF2023-4, pp. 17-22, (2023).
[2] 吉田裕紀, 中西知嘉子, “アクセラレート回路を用いたエッジ AI のデータ共有方法の検討”, 電子情報通信学会総合大会(2024).
[3] Ultra96-V2, https://japan.xilinx.com/products/boards-and-kits/1-vad4rl.html
[4] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, Jian Sun, “YOLOX: Exceeding YOLO Series in 2021”, arXiv preprint arXiv:2107.08430 (2021)
[5] Vitis HLS, https://japan.xilinx.com/support/documentation-navigation/design-hubs/2020-2/dh0090-vitis-hls-hub.html
[6] Vivado Design Suite, https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals_j/xilinx2020_2/ug910-vivado-getting-started.pdf
[7] 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”, 電子情報通信学会総合大会(2022).
[8] 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", 電子情報通信学会ソサイエティ大会(2021).
[9] Kria KV260, https://www.amd.com/ja/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html
[10] Vitis AI, https://japan.xilinx.com/products/design-tools/vitis/vitis-ai.html

† 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