FPGA AIハッカソン 参加記

東京大学 中村・高瀬研究室

D2 齋藤真

目次

- 自己紹介・参加目的
- FPGAハッカソン概要
- 研究室内運営
- 実装方針
- 改善手法1：YOLOの改善
- 改善手法2：ソフトウェア高速化
- 改善手法3：ハードウェア高速化

目次

FPGA AI デザインハッカソン紹介

- 概要
  - 東京大学『Agile-X～革新的半導体技術の民主化拠点』プロジェクト
    - https://www.agile-x.t.u-tokyo.ac.jp/project/FPGAAIhackathon/
  - 今年はHotSPA@猪苗代湖 で開催された
- 目的
  - 東大Agile-X 革新的半導体技術の民主化拠点プロジェクトによる半導体設計ハッカソンを一般に試行することで、半導体への興味を持つ学生を増やす。
- その他
  - YoLo-v3 を FPGA ボード V260に実装し、性能と精度を競う。高校生から修士大学院生までを想定。

FPGA AI デザイン
ハッカソンへのお誘い

目的

本デザインハッカソンは、東大 Agile-X 革新的半導体技術の民主化拠点プロジェクト（https://www.agile-x.t.u-tokyo.ac.jp/）による半導体設計ハッカソンを一般に試行することで、半導体への興味を持つ学生を増やすことを目的とします。

テーマ

YoLo-v3 を FPGA ボード KV260 に実装し、性能と精度を競います。具体的な内容については Agile-X の Web サイトを参照ください。

連絡先

ハッカソン実行担当　天野英晴
hunga@dlab.t.u-tokyo.ac.jp
情報は、随時
https://www.agile-x.t.u-tokyo.ac.jp/hackathon に掲示します。

方法

① 参加者は高校生から修士大学院生までを想定しています。本ハッカソンは、ハードウェア設計の経験が少ない学生を対象としますが、プロの参加を拒みません。1－3 人でグループを組み **Google Form(4 月 1 日にオープン)より参加登録を 4 月中**に行います。先着 16 グループで締め切ります。
② 設計用データの入った KV260 ボードとケーブル類が連休明けに送付されてきます。高額なものではないですが、国家プロジェクトで購入したボードですので誠意をもって大切に管理してください。ホスト PC（Windows PC）は各自ご用意ください。実行環境を後に Web に掲示する方法に従ってセットアップしてください
③ 参加者は、公開されている Wiki（https://github.com/takgto/utokyo-chipathon2023/wiki）に従って YoLo-v3 を動かして、動画の認識を行ってください。その後、ヒントを見ながら最適化してください。
④ **6 月 10 日、電子情報通信学会、情報処理学会連合研究会 HotSPA の会場**で本番動画に対して認識を実行していただきます。HotSPA へ参加できない方は事前にデータを送っていただき、運営側で実行します。
⑤ 高い性能を良い精度で実現したグループを表彰します。
⑥ 貸与した器材は、会場で回収します。参加できなかった方は、返送をお願いします。

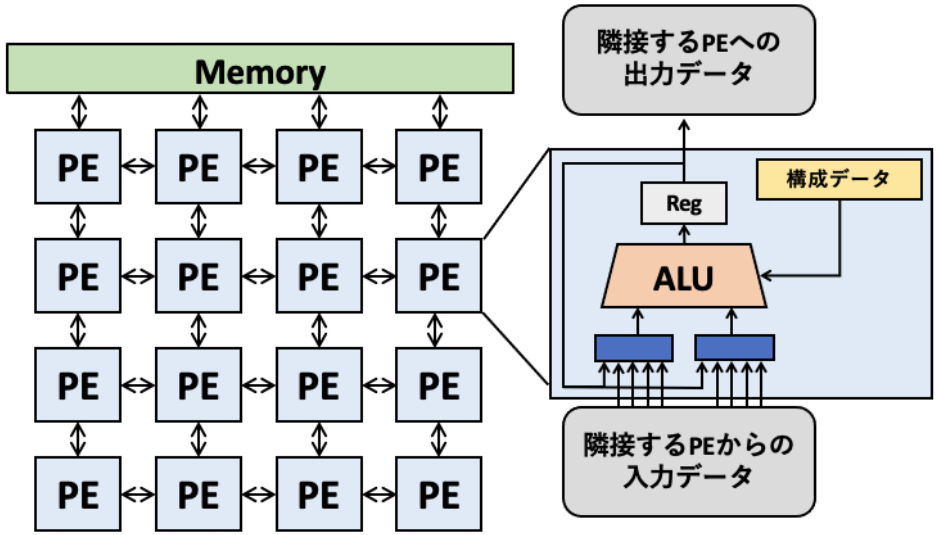
2025.3.10

発表者紹介

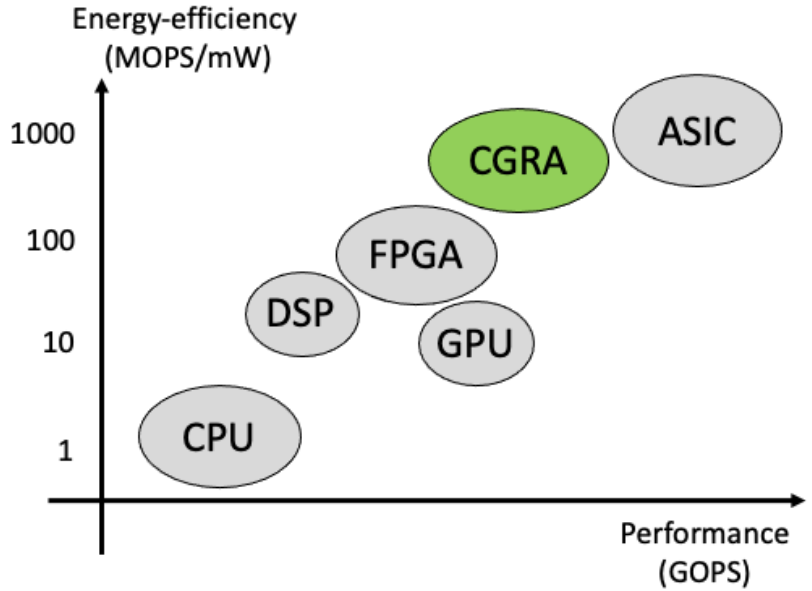
- 齋藤 真
- 経歴
  - 東京大学 工学部 計数工学科システム情報工学コース
    - 学部研究：ドローン制御手法
  - 修士・博士：東京大学情報理工学系研究科 システム情報学専攻 中村・高瀬研
    - 修士研究：CGRA向けコンパイラ手法
    - 博士研究：再構成可能アーキテクチャを用いたロボティクス向け計算機システムの創出 (DC1)
- FPGA経験は？
  - 学部は制御手法・信号処理等も学ぶ，FPGAは触れる程度
  - 一度FPGA + YOLOにチャレンジしたが挫折
  - 研究はAIアクセラレータ向けコンパイラ，HDLは多少書ける

研究テーマ：CGRA (粗粒度再構成可能アーキテクチャ)

- 2次元配列状に Processing Element (PE) を持つ領域特化アーキテクチャ
  - PEで実行する命令・PE間の接続を構成データで設定 ⇨ 動的にハードウェアを再構成可能
- CPU・GPU・FPGA等に比べて計算性能・消費電力で優れる [5]
- 粗粒度のため構成データが少ない ⇨ 構成データの書き換えが速い



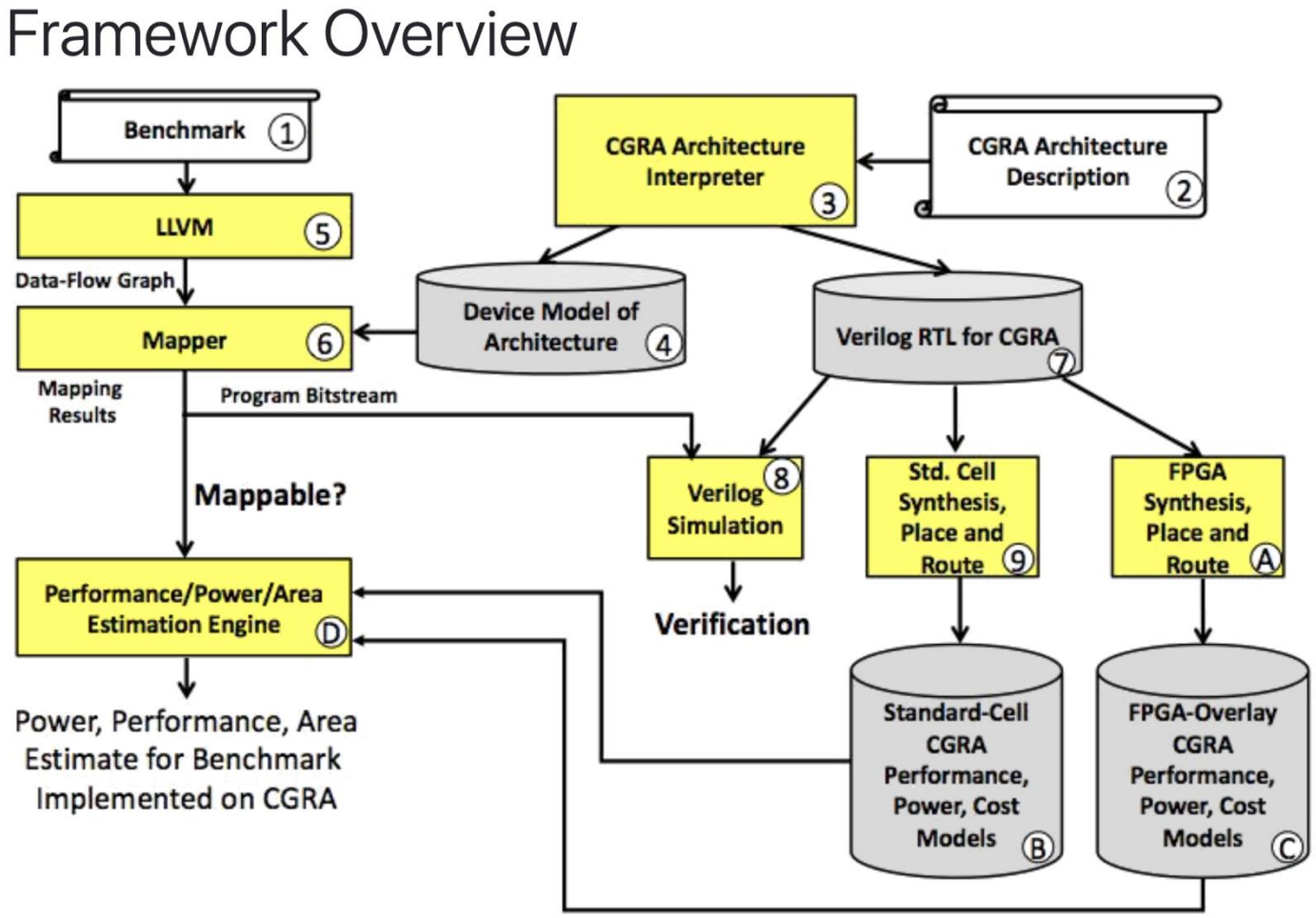
CGRAの構成



他アーキテクチャとの性能比較 [1]

研究テーマ：CGRA (粗粒度再構成可能アーキテクチャ)

- CGRAは実チップが流通していない ⇨ FPGAを用いた評価も行われる



参加目的

- 目的：AIアクセラレータ研究者として深層学習の評価を行えるようにする
  - 自分で勝手にやれば良くない，，？
- なぜFPGA開発に手をつけられていなかったのか？
  - 原因①：FPGAを学ぶハードルの高さ
    - ハードウェア環境への依存が強い
  - 原因②：何を目標にして開発を進めるか決めにくい

例：学部4年でFPGAで挫折した体験

- 状況
  - 体験配属された研究室でYOLOをFPGA上で動かすことにチャレンジした
- どこで詰まったか？
  - どのハードウェア使えばよい？
    - 最初にUltra96v2を使用．先行例が少なく他のFPGAを検討した
  - 何で実装すれば良い？
    - 高位合成？Vitis-AIを利用する？FPGAを制御するOSはどれを使う？
    - Caffeで実装された学習済みYOLOv3の量子化・コンパイルをFPGA側で読み込むのに苦戦

参加目的・ハッカソンの利点

- 目的：AIアクセラレータ研究者として深層学習の評価を行えるようにする
  - 自分で勝手にやれば良くない，，？
- なぜFPGA開発に手をつけられていなかったのか？
  - 原因①：FPGAを学ぶハードルの高さ
    - ハードウェア環境への依存が強い
  - 原因②：何を目標にして開発を進めるか決めにくい
- ハッカソンを活用した理由
  - 利点①：チーム参加なので課題を複数人で共有しやすい
  - 利点②：動くところまでは運営側でサポートしてくれる
    - ベースライン実装が既に与えられているのは楽！
  - 利点③：対象アプリケーションや評価指標が与えられるので，開発計画が立てやすい

目次

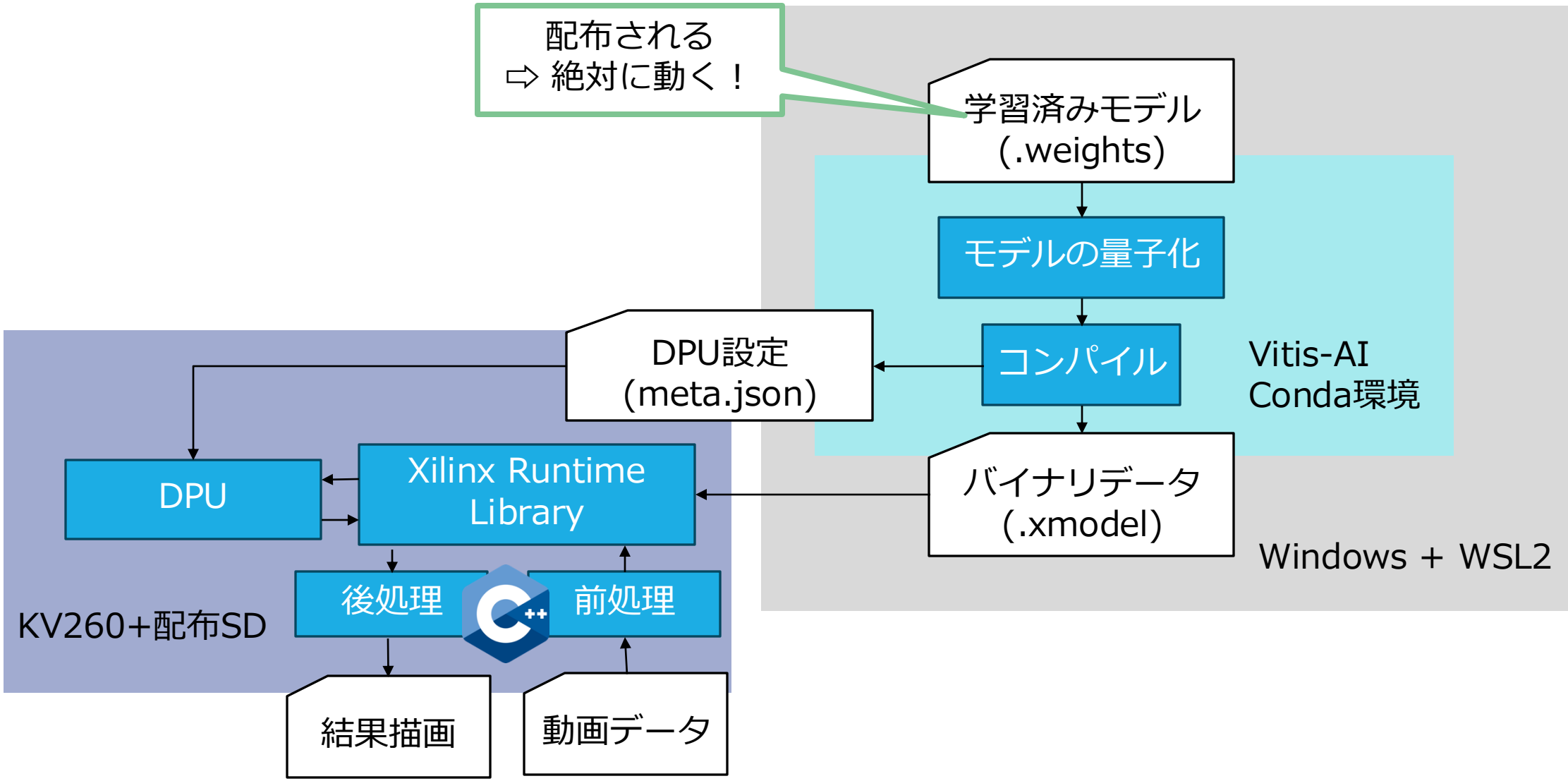
FPGAハッカソン詳細ルール

- ルール
  - 1チーム3人、KV260ボードを利用して画像認識の精度を競う
  - FPS × mAP のスコアを争う ⇨ 要するに速度×精度
    - 認識するのは人・車・自転車の3種類
- サポート環境
  - KV260ボード・周辺機材の貸し出し
  - チュートリアル・初期ソフトウェア
    - YOLOv3の学習済みモデルが配布
    - ふんがのブログ (https://hunga.sblo.jp/)
    - チュートリアル (https://github.com/takgto/utokyo-chipathon2023/wiki)
  - ハッカソン用Slackでの質問・相談が可能

KV260ってどんなボード？

- K26 SOM (Zynq UltraScale+ MPSoC)+ 画像認識用のインタフェースが豊富

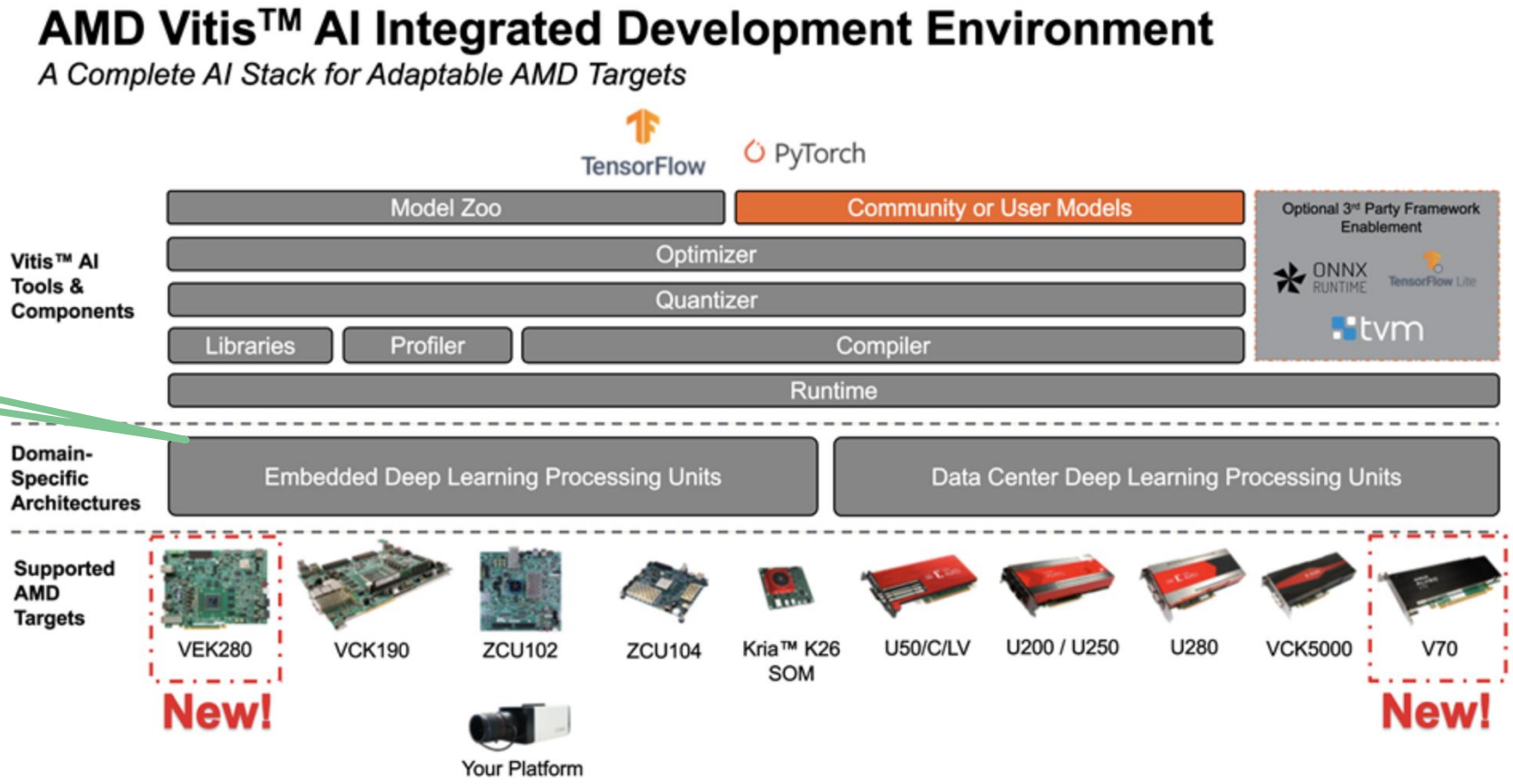
提供されたベースライン実装の全体像



Vitis-AIとは？

- AMD ハードウェア プラットフォーム上での AI 推論を高速化する環境
  - FPGAのPL部分の複雑さが緩和、深層学習推論アプリケーションの開発が容易に

DPUで抽象化
⇨ これをチューニング



目次

- 自己紹介・参加目的
- FPGAハッカソン概要
- 研究室内運営
- 実装方針
- 改善手法1：YOLOの改善
- 改善手法2：ソフトウェア高速化
- 改善手法3：ハードウェア高速化

運営方針

- 研究室から3チームが参加
  - 実装・ツールの使い方などは共通チャンネルでみんなで解決
    - 例) MacOSでチュートリアルを動かす方法, VSCodeからRemote SSHできない, etc.
  - チュートリアルをみんなで動かしてみる作業会の開催
  - 各チームの実装方針の相談等は個別チャンネルで
    - FPGAに対して知識がある教員・学生がそれぞれのチャンネルに参加する

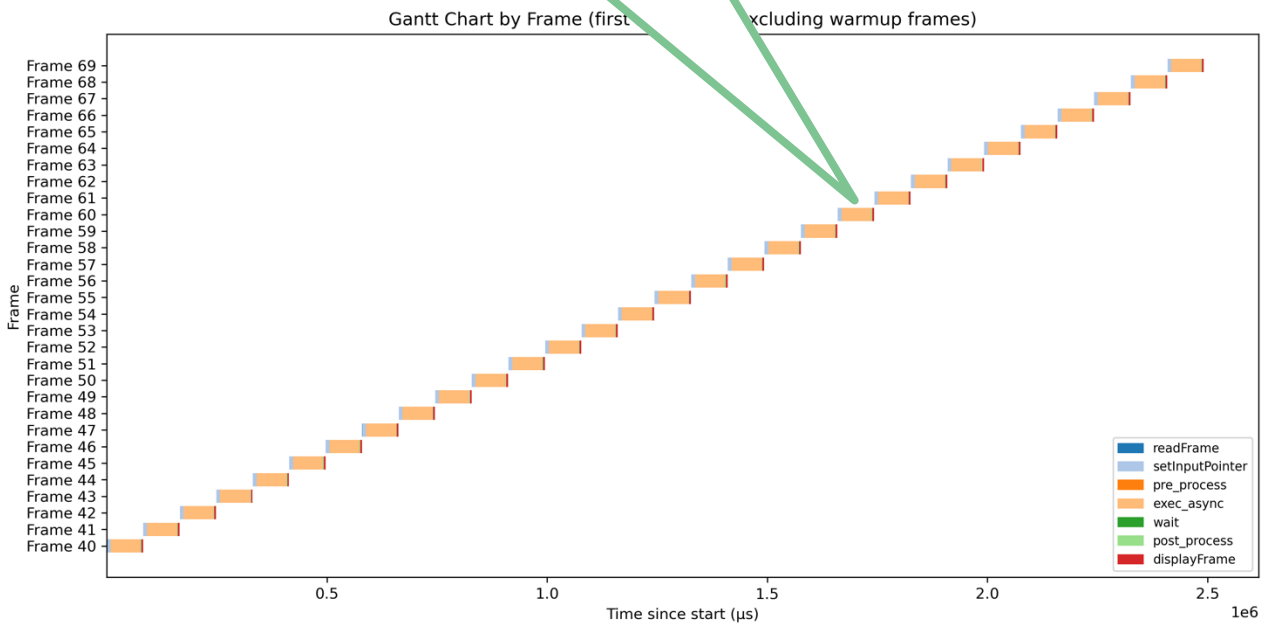
目次

- 自己紹介・参加目的
- FPGAハッカソン概要
- 研究室内運営
- 実装方針
- 改善手法1：YOLOの改善
- 改善手法2：ソフトウェア高速化
- 改善手法3：ハードウェア高速化

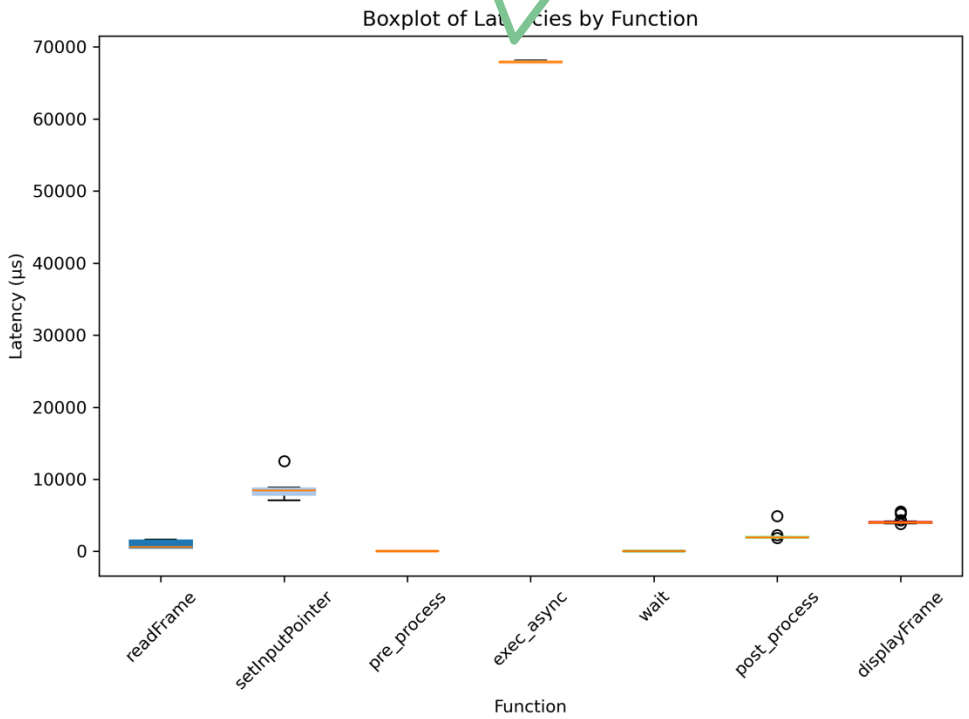
事前評価

- とりあえずベースライン実装を動かしてみる

シングルスレッドなので
FPSは低い



推論部分がボトルネック



先行研究

- FPGA-Accelerated YOLOX With Enhanced Attention Mechanisms for Real-Time Wildfire Detection on AAVs (IEEE TIM 2025)
  - KV260を用いたYOLOの高速化を実行
  - FPGAの消費リソースやDPUパラメータが記載されている、再現実装の指標の1つに

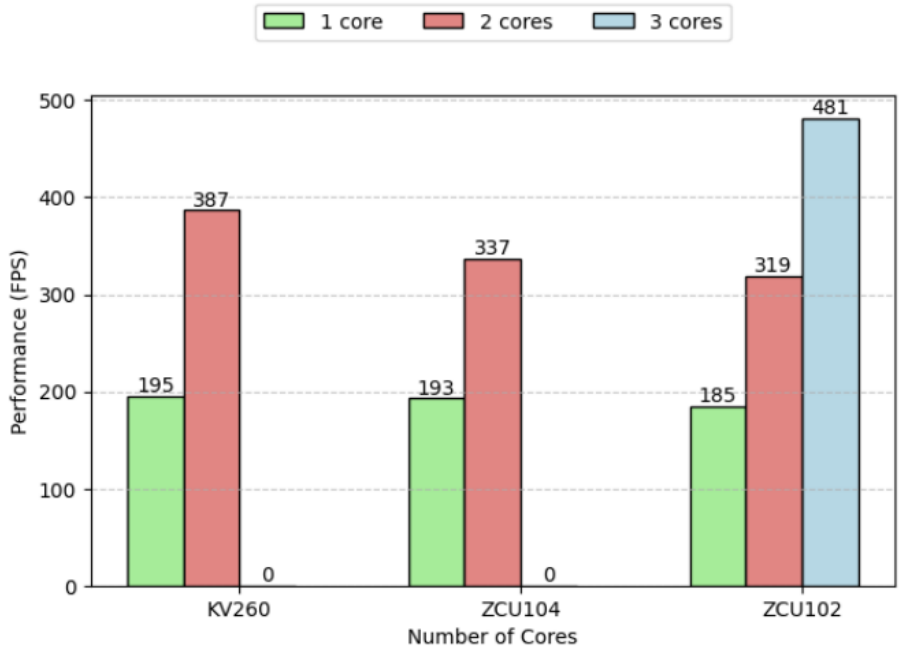


Fig. 6. Performance of FPGA devices with varying DPU configurations.

TABLE VI
FPGA RESOURCE UTILIZATION ACROSS DIFFERENT BOARDS AND CORE CONFIGURATIONS

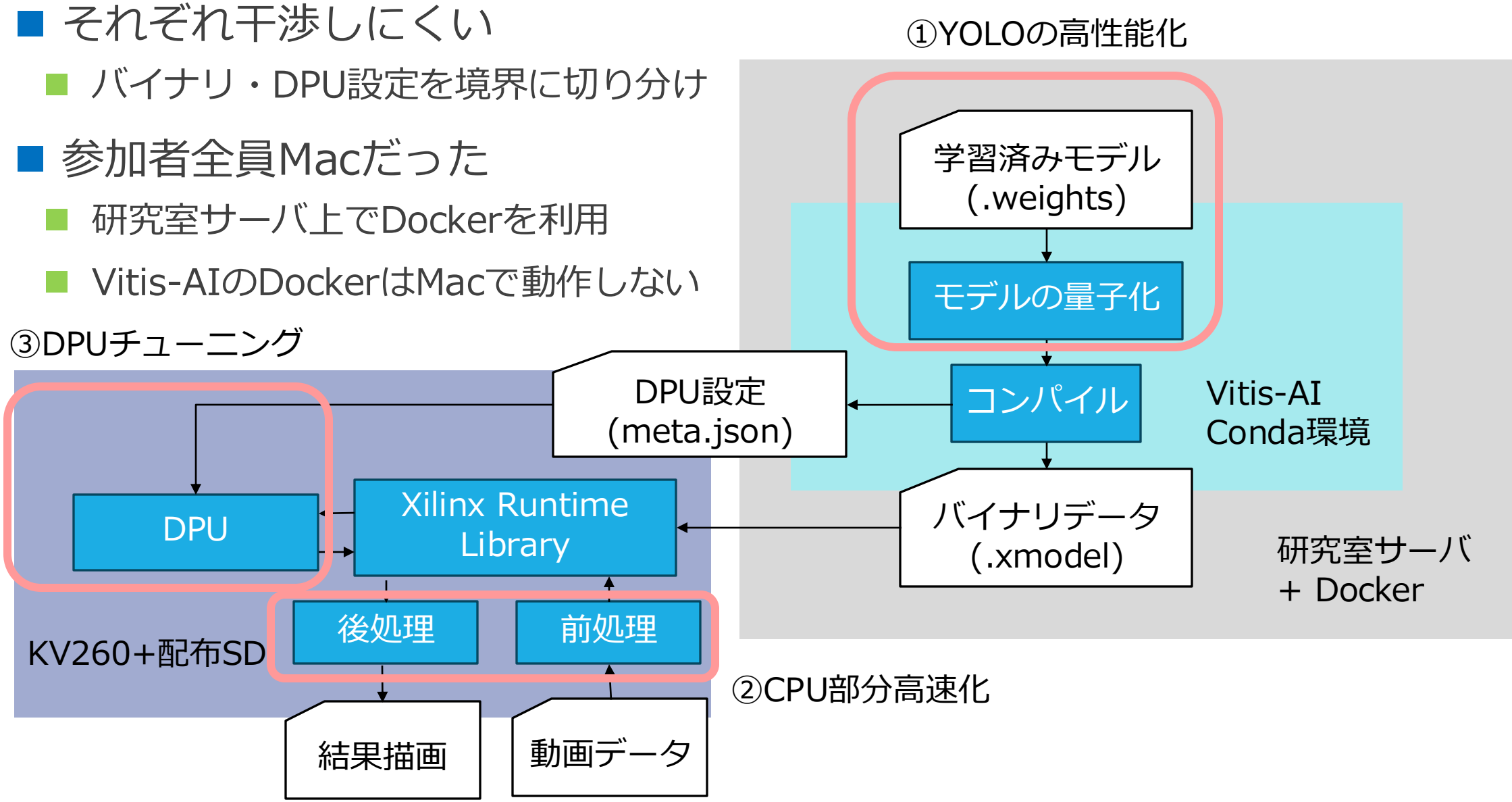
Boards	LUT (Used/Available)	FF (Used/Available)	BRAM (Used/Available)	URAM (Used/Available)	DSP (Used/Available)	LUTRAM (Used/Available)
ZCU102						
1-core	52,161/274,080 (**19.03%**)	98,249/548,160 (**17.92%**)	255/912 (**27.96%**)	–	710/2,520 (**28.17%**)	5,647/144,000 (**3.92%**)
2-core	107,237/274,080 (**39.12%**)	187,663/548,160 (**34.24%**)	512/912 (**56.14%**)	–	1,451/2,520 (**57.58%**)	12,344/144,000 (**8.57%**)
3-core	165,111/274,080 (**60.24%**)	302,294/548,160 (**55.15%**)	769/912 (**84.32%**)	–	2,138/2,520 (**84.84%**)	21,662/144,000 (**15.04%**)
ZCU104						
1-core	63,003/230,400 (**27.35%**)	107,833/460,800 (**23.40%**)	259/312 (**83.01%**)	–	718/1,728 (**41.55%**)	–
2-core	95,724/230,400 (**41.55%**)	188,110/460,800 (**40.82%**)	154/312 (**49.36%**)	96/96 (**100%**)	718/1,728 (**41.55%**)	–
KV260						
1-core	65,139/117,120 (**55.62%**)	108,532/234,240 (**46.33%**)	19/144 (**13.19%**)	42/64 (**65.62%**)	546/1,248 (**43.75%**)	–
2-core	84,511/117,120 (**72.15%**)	167,544/234,240 (**71.52%**)	57/144 (**39.58%**)	64/64 (**100%**)	918/1,248 (**73.56%**)	–

ハッカソン側の実装方針案

- ハッカソンのWikiには以下の方針が示された
  - **1. YOLOv3をYOLOv3-tinyに変更する**
  - **2. パイプライン化マルチスレッド化**
  - **3. 入力画像サイズを変えてみる**
  - 4. KerasモデルをONNXに変更、最適化
  - **5. DPUのチューニングを行う**
  - 6. Verilogを書く

チーム開発の役割分担

- それぞれ干渉しにくい
  - バイナリ・DPU設定を境界に切り分け
- 参加者全員Macだった
  - 研究室サーバ上でDockerを利用
  - Vitis-AIのDockerはMacで動作しない



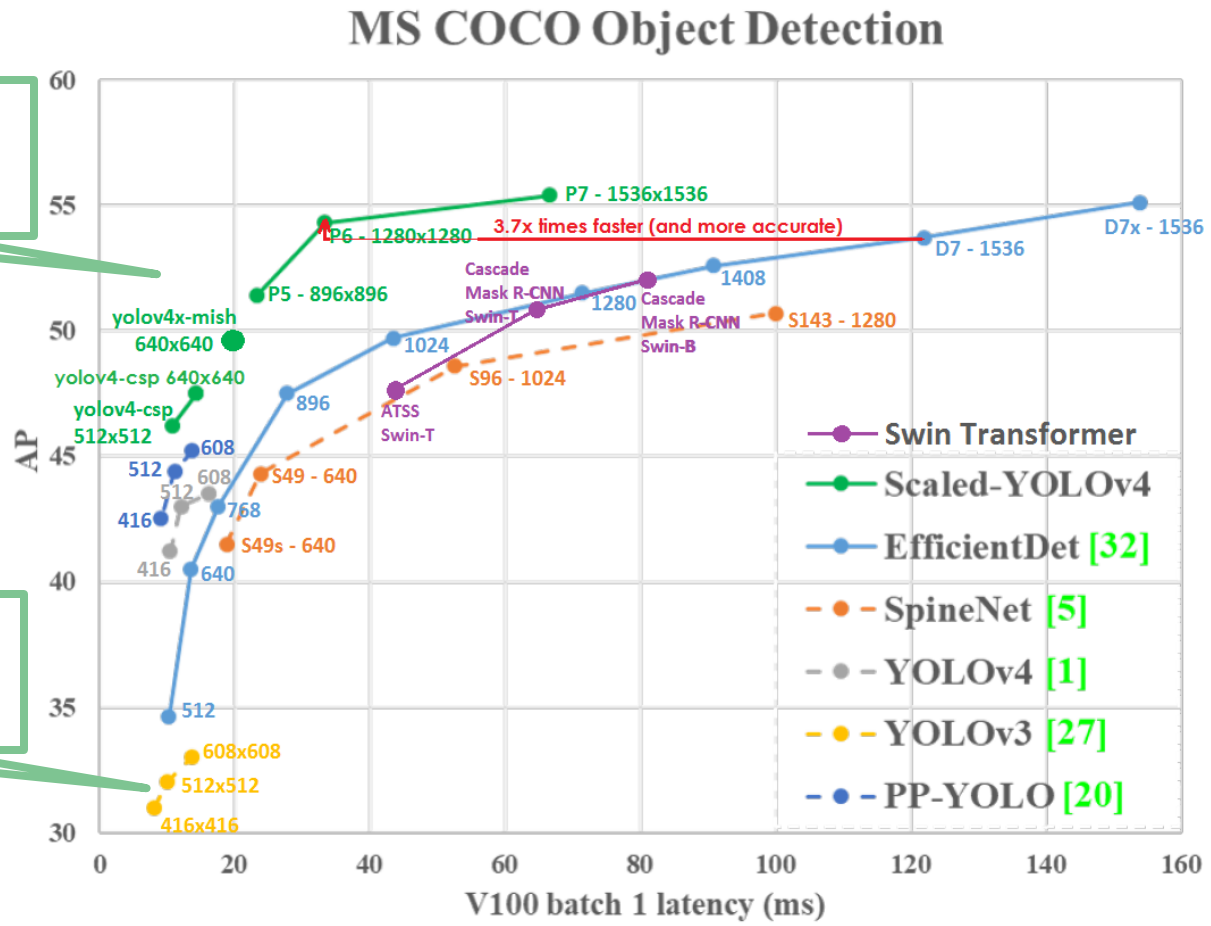
目次

YOLOとは？

- You Only Look Once：物体検出と物体認識を同時にできる

上の方が良さそう

YOLOv3
⇨ 精度低くない？



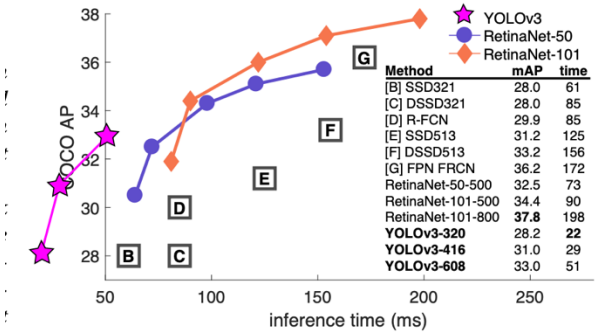
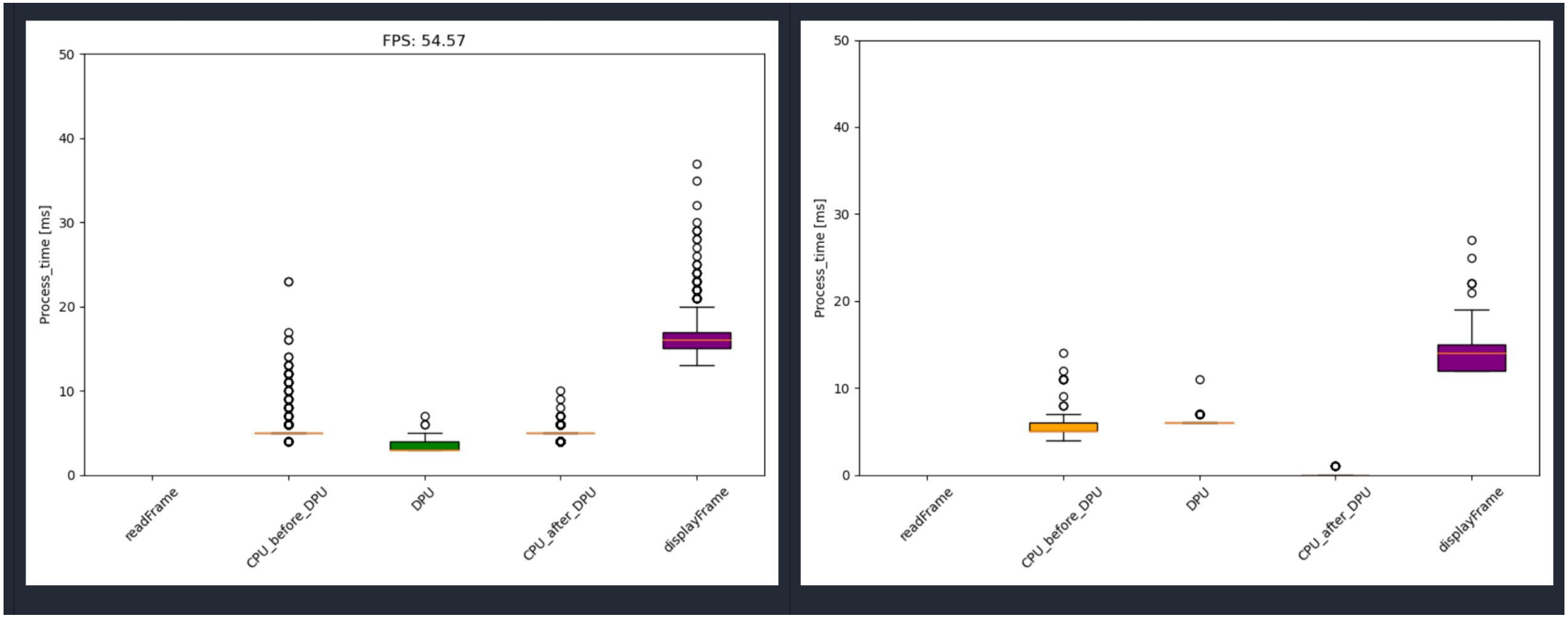


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

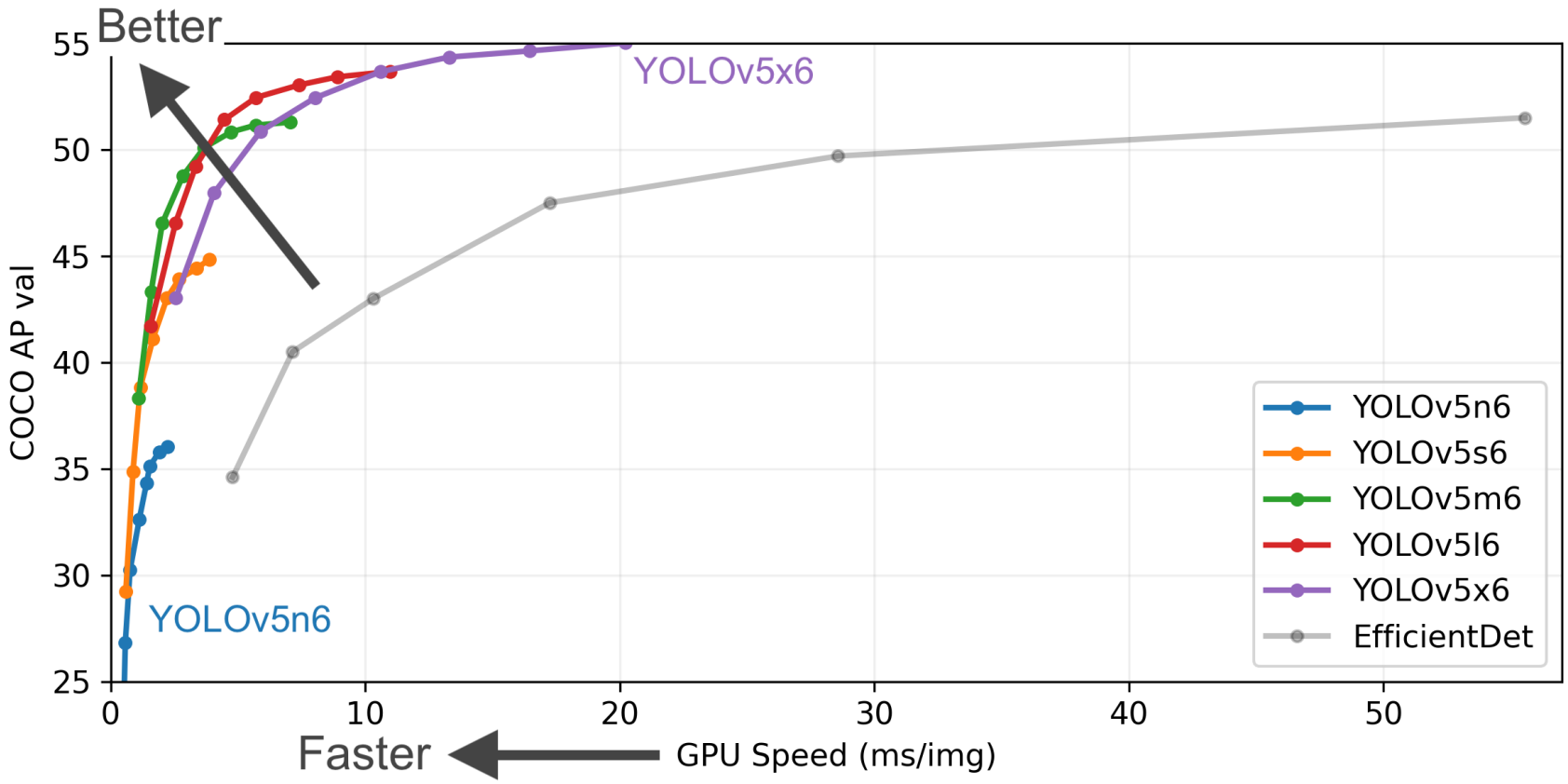
YOLOv5による高速化

- YOLOv5n vs. YOLOv3-tiny
  - YOLOv5の活用によって処理時間が短縮



YOLOv5による高速化

- YOLOv5n vs. YOLOv3-tiny
  - YOLOv5の中でもサイズ選択が可能、速度と精度のトレードオフ



その他の工夫

- 学習データの改善
  - 提供されたのはCOCOデータセット 5000枚 ⇨ Open Images Dataset v7 80000枚を利用
  - YOLOv11で擬似ラベルを作成、入力画像を正規化したのちに416×416 にリサイズ ⇨ mAP 45%
- 枝刈り
  - 全パラメータの約30% を枝刈り ⇨ mAP 42%
- 量子化
  - int8 量子化を適用 ⇨ mAP は34% となった.

目次

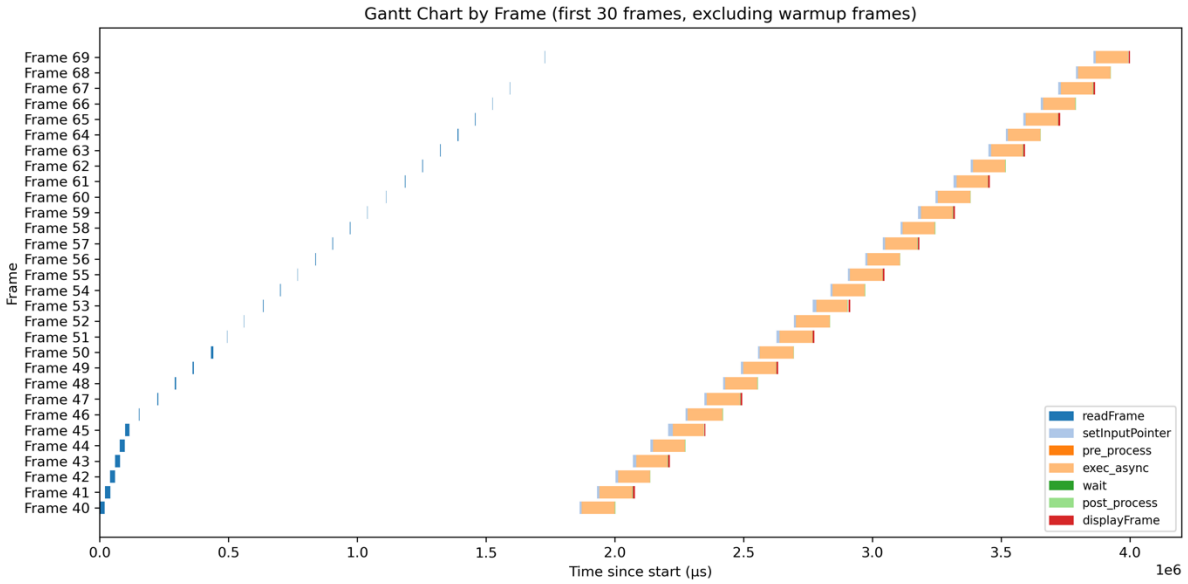
マルチスレッド+パイプライン化

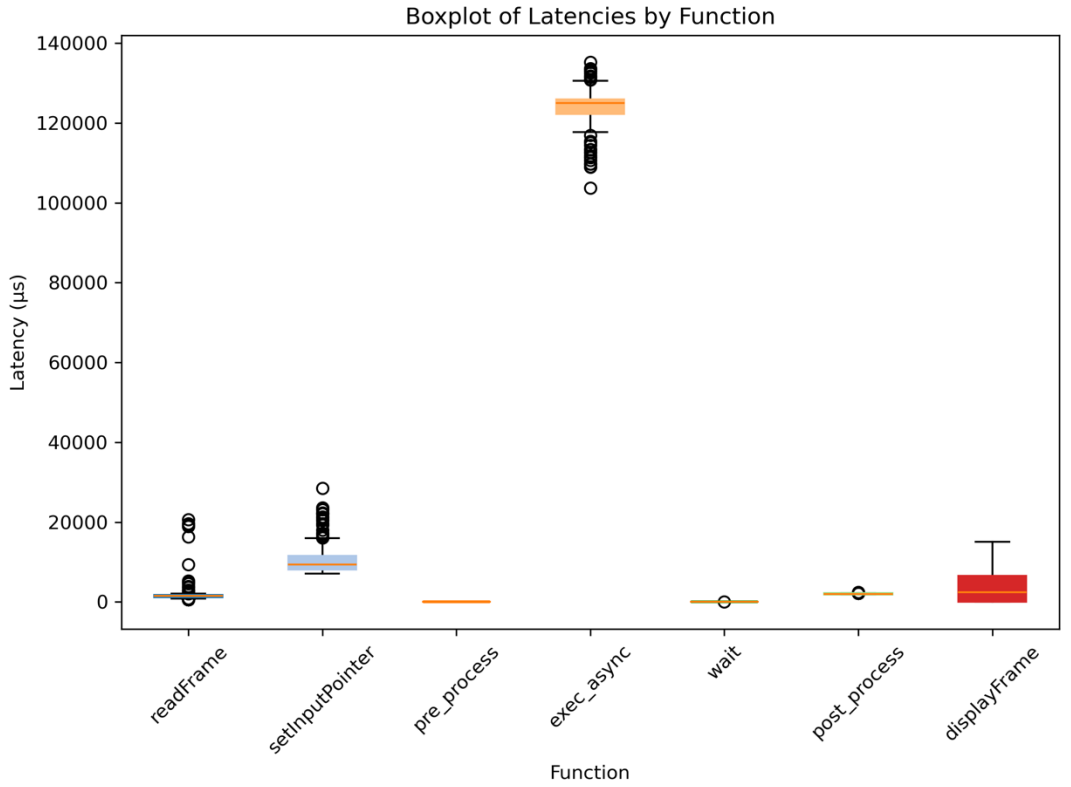
- C++のThreadライブラリを用いてマルチスレッド化した
  - とりあえず前処理・後処理で1スレッドずつ、YOLOを2スレッド

```
array<thread, 4> threadsList = {
    thread(readFrame, argv[2], ref(fr)),
    thread(displayFrame, ref(shw)),
    thread(runYOLO, runner.get(), ref(fr), ref(shw)),
    thread(runYOLO, runner1.get(), ref(fr), ref(shw)),
};
```

マルチスレッド+パイプライン化

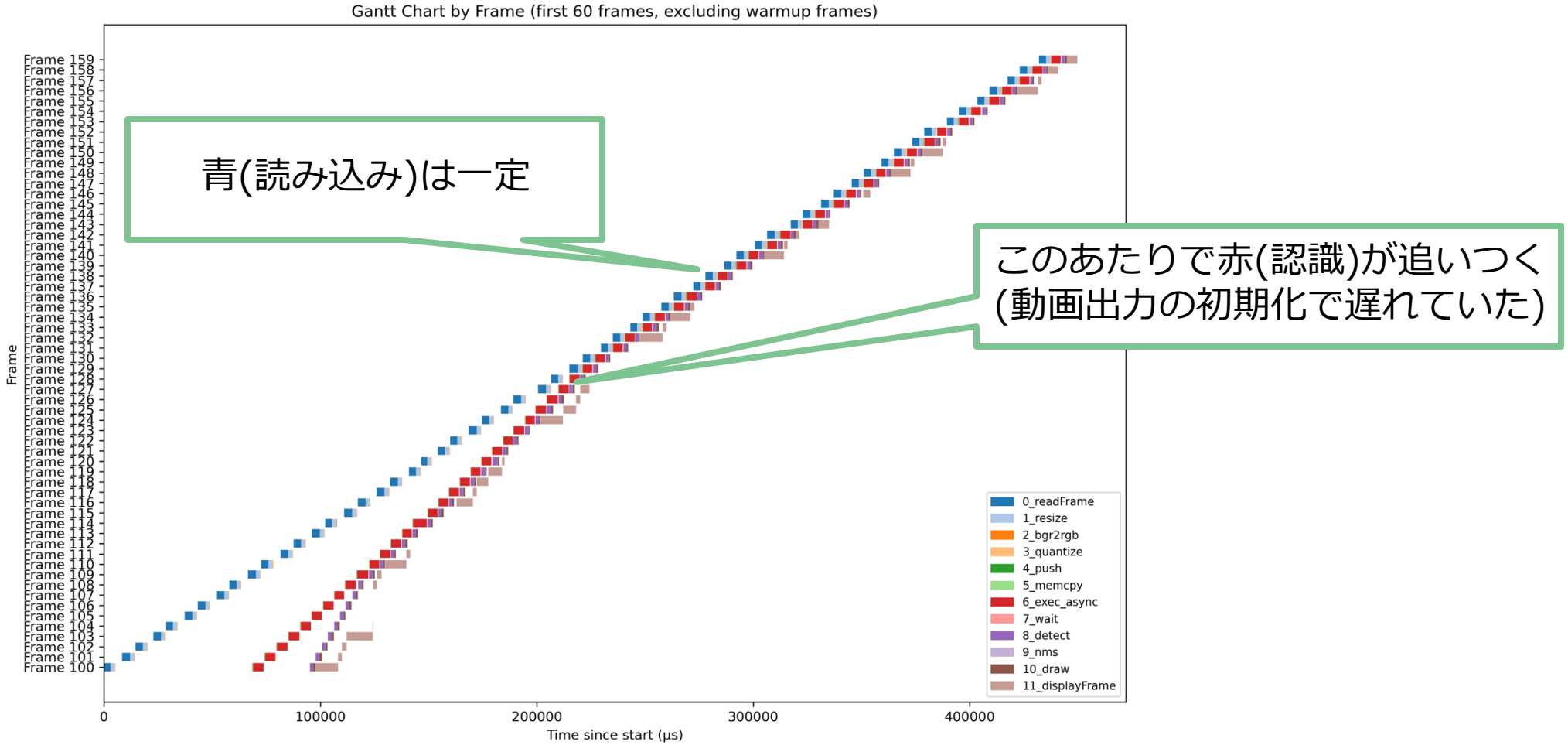
- C++のThreadライブラリを用いてマルチスレッド化した
  - とりあえず前処理・後処理で1スレッドずつ、YOLOを2スレッド





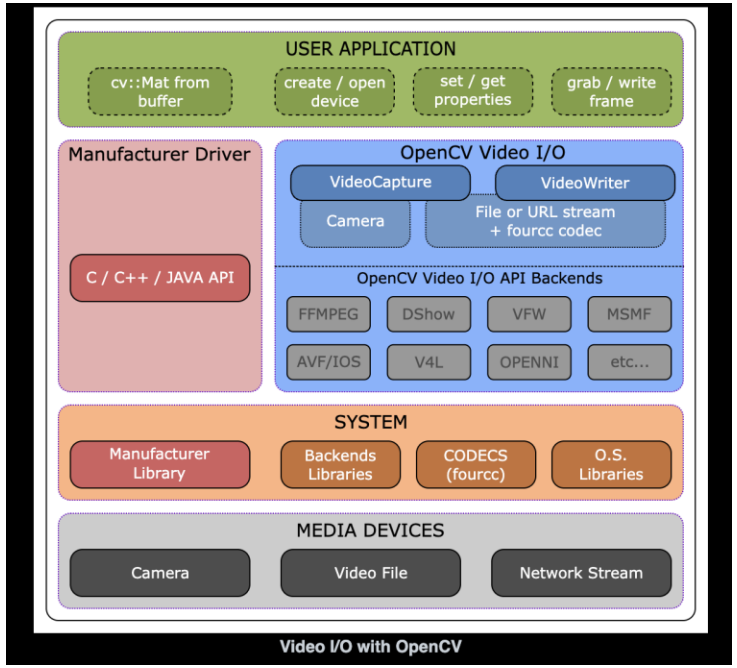
IOの高速化

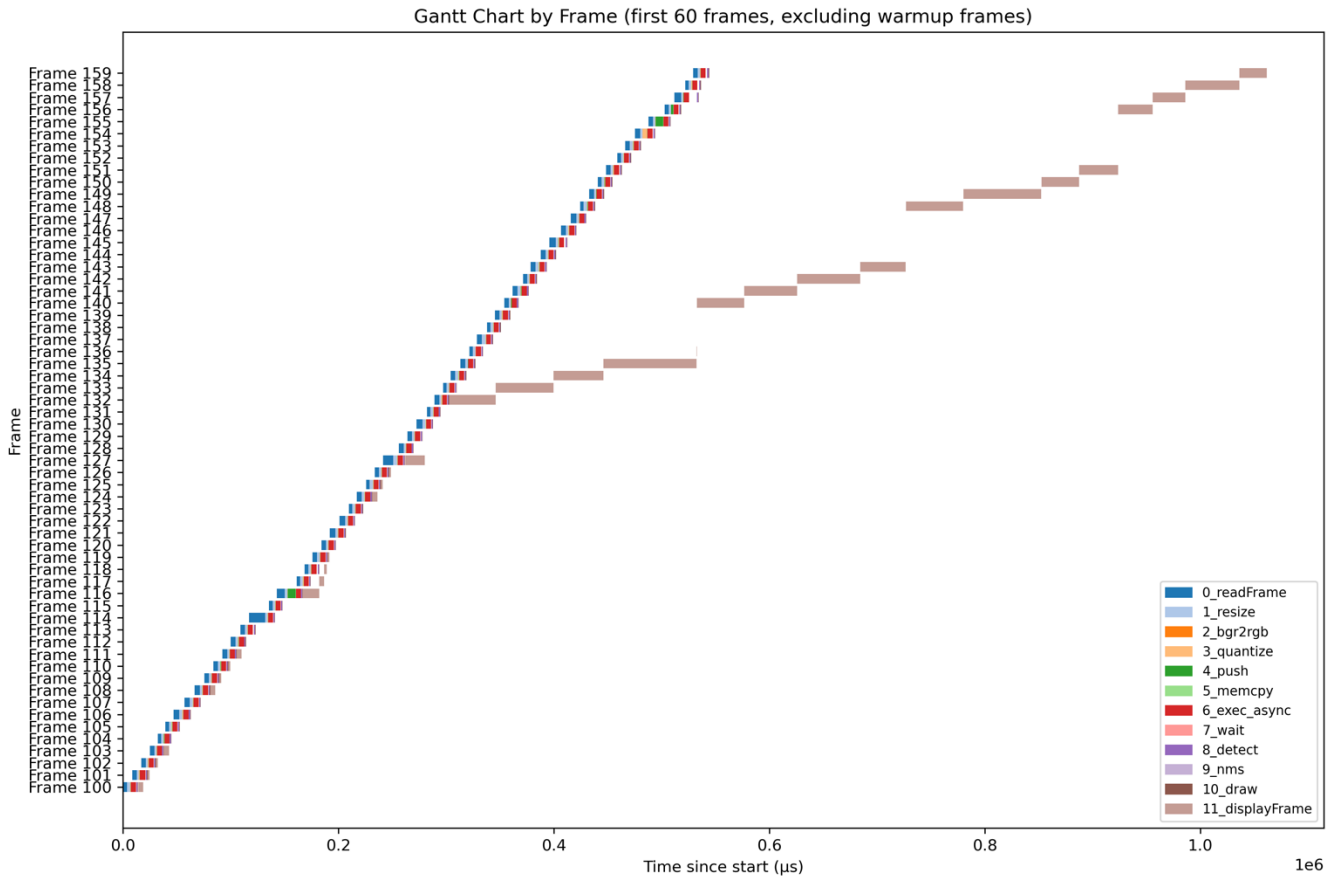
- 動画読み込みがボトルネックになる



IOの高速化

- 動画のデータ形式がwebm形式なのも問題
- OpenCV のVideo IOのバックエンドを変更
  - FFMPEGを使用
  - ストリーム処理ができるようになると良い
    - その中でもいくつか選別した





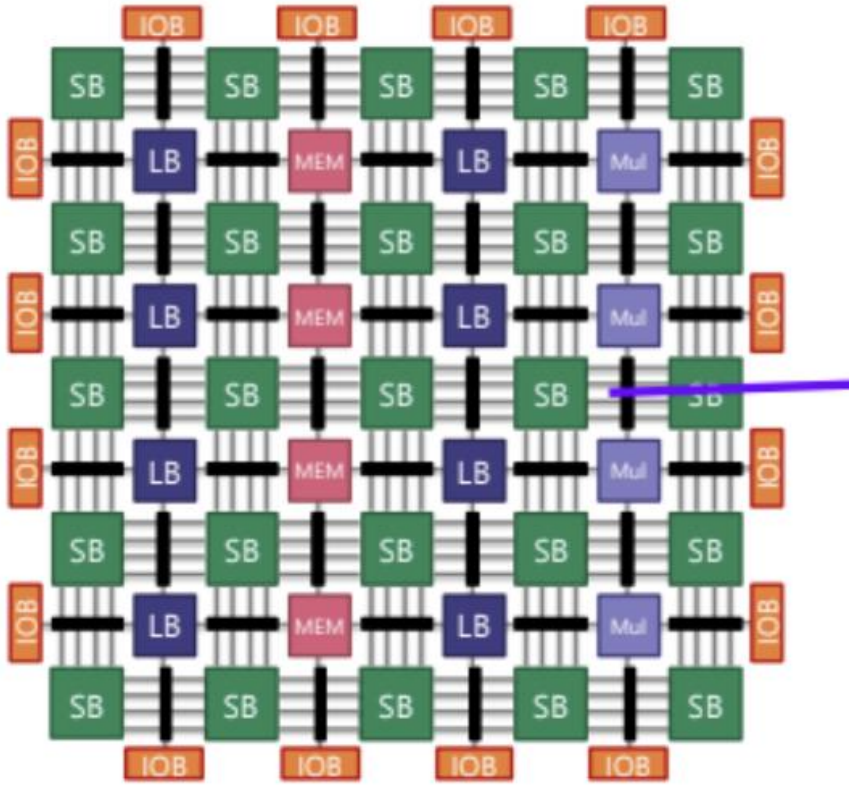
目次

ハードウェア高速化の方針

- 方針1：YOLOを処理する部分を2つに増やす
- 方針2：YOLOを処理する部分をチューニングする

FPGAって何？

- 任意の論理回路を実現することが可能
  - 多数のLUT (ルックアップテーブルを用いる)
  - その他、レジスタ・RAM・DSPなどを用いて再構成する



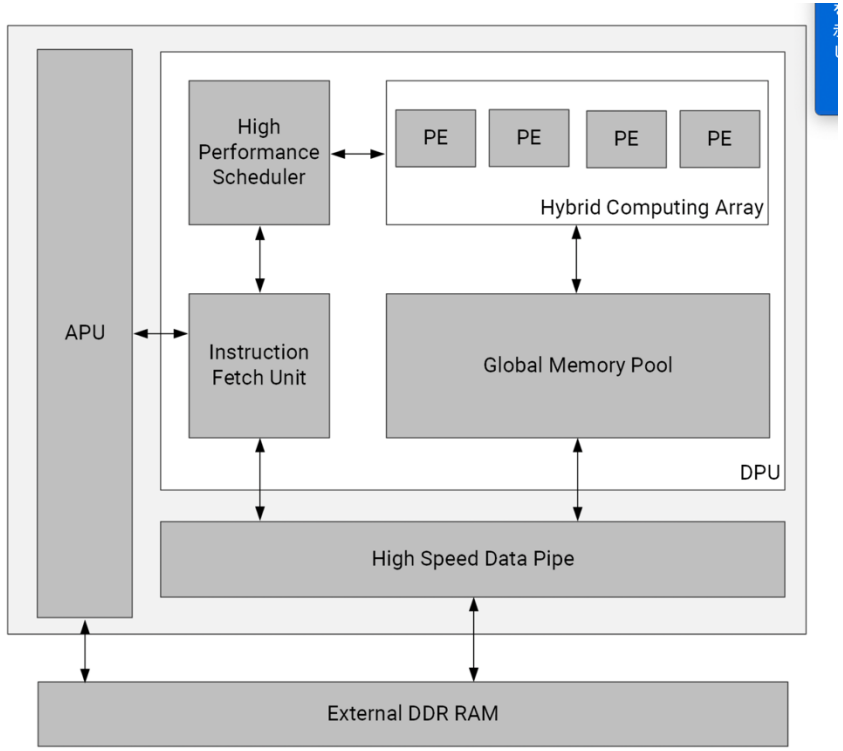
DPUのハードウェア構成

- DPUCZDX8Gアーキテクチャが使用可能
  - リソースの異なる複数DPUを選択して実装することが可能

Displayed in the header

DPUCZDX8G アーキテクチャ	LUT	レジスタ	ブロック RAM	DSP
B800	29721	41147	90	166
B1024	34074	48057	104	230
B1152	32169	47374	121	222
B1600	38418	58831	126	326
B2304	42127	68829	165	438
B3136	46714	79710	208	566
B4096	52161	98249	255	710

もう 1 つの DPUCZDX8G シングル コア プロジェクトのサンプルは、ZCU104 プラットフォームに基づきます。このプロジェクトでは、イメージおよび重みバッファーで UltraRAM が使用されています。また RAM 使用量を低、チャネル拡張、alu parallel = PP/2、conv: leaky ReLU + ReLU6、alu: ReLU6 を選択し、DSP 使用量を低に設定しました。このプロジェクトのリソース使用状況を次に示します。

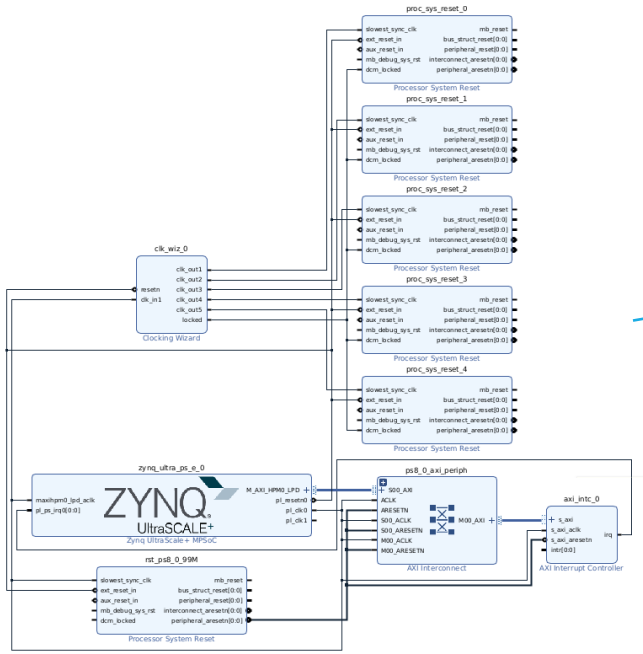


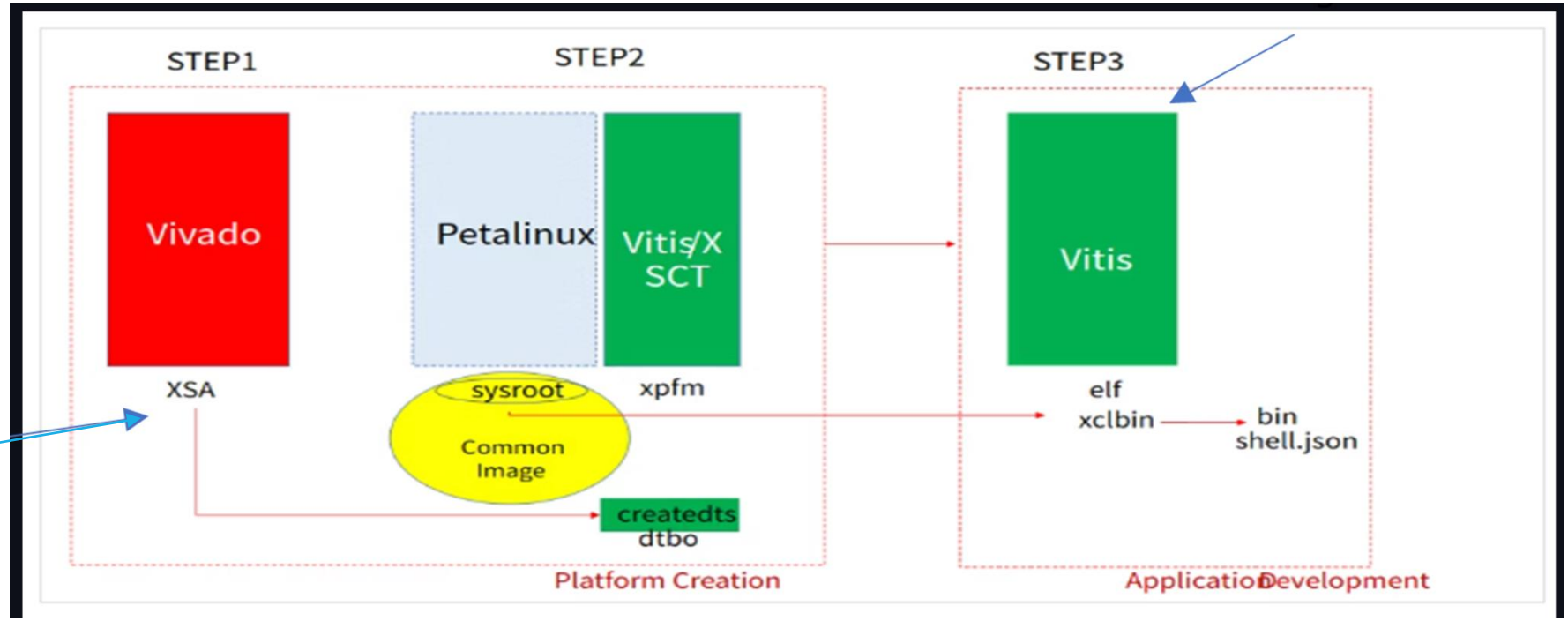
DPUのハードウェア構成

- DPUCZDX8Gアーキテクチャが使用可能
  - URAMの割り当て (IMG/WGT/BIAS)
    - デフォルトはBRAM ⇨ より大きいURAMに変更
  - DRAMの割り当て (IMG/WGT/BIAS)
    - LUTが余っていた時、DRAMとして使用可能
  - RAMの利用率 (HIGH / LOW)
  - DSPの利用率 (HIGH / LOW)
  - RELUのタイプ

Site Type	Used	Fixed	Prohibited	Available	Util%
CLB LUTs*	51740	0	0	117120	44.18
LUT as Logic	44603	0	0	117120	38.08
LUT as Memory	7137	0	0	57600	12.39
LUT as Distributed RAM	2596	0			
LUT as Shift Register	4541	0			
CLB Registers	99594	0	0	234240	42.52
Register as Flip Flop	99594	0	0	234240	42.52
Register as Latch	0	0	0	234240	0.00
CARRY8	1279	0	0	14640	8.74
F7 Muxes	2520	0	0	58560	4.30
F8 Muxes	0	0	0	29280	0.00
F9 Muxes	0	0	0	14640	0.00

Vivado・Vitisでの開発フロー





FPGAでのDPU切り替え

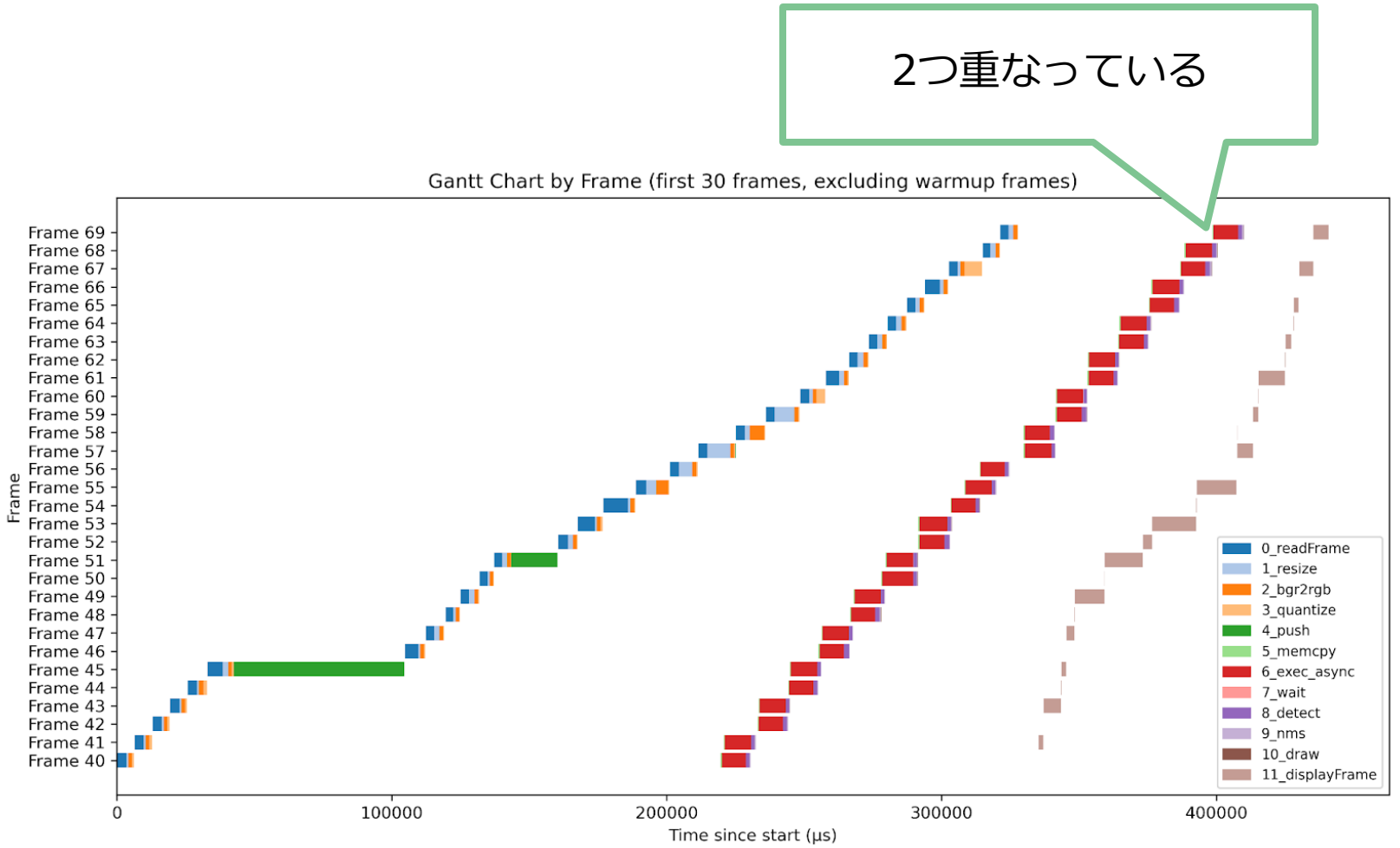
- FPGA上で複数のDPU設定を置いておくことができる
  - Petalinux側でコマンドラインで切り替えながら実験することが可能
  - 先行研究ではB4096が2つ乗っていたが、そこまで至らず、、
  - B4096 1コア vs. B1024 2コア

```
petalinux@xilinx-kv260-starterkit-20222:~$ sudo xmutil listapps
                  Accelerator          Accel_type                          Base          Base_type      #slots(PL+AIE)         Active_slot

                       1024x2            XRT_FLAT                       1024x2           XRT_FLAT               (0+0)                  -1
                       1152x2            XRT_FLAT                       1152x2           XRT_FLAT               (0+0)                  -1
                         2304            XRT_FLAT                         2304           XRT_FLAT               (0+0)                  -1
                         4096            XRT_FLAT                         4096           XRT_FLAT               (0+0)                  -1
                    4096_300Hz            XRT_FLAT                    4096_300Hz           XRT_FLAT               (0+0)                  -1
                       4096_v2            XRT_FLAT                       4096_v2           XRT_FLAT               (0+0)                  -1
```

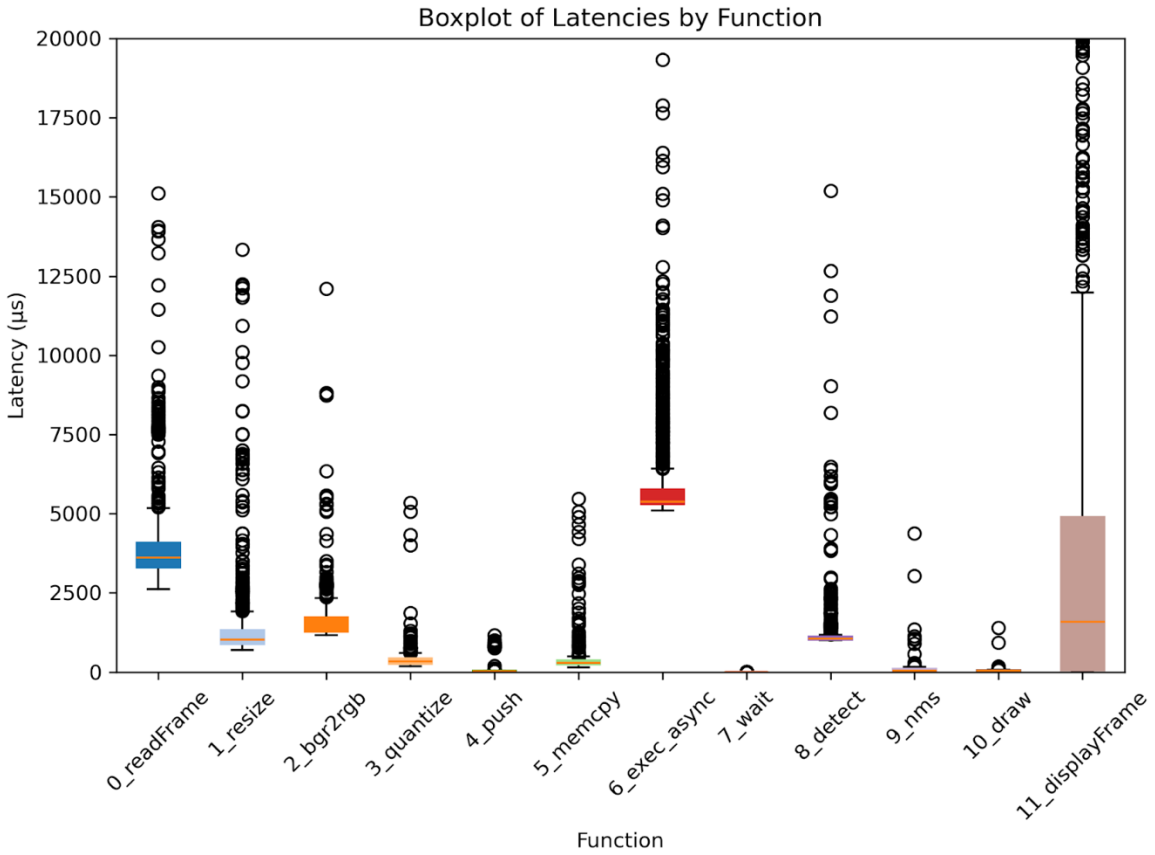
2 Coreでの並列処理

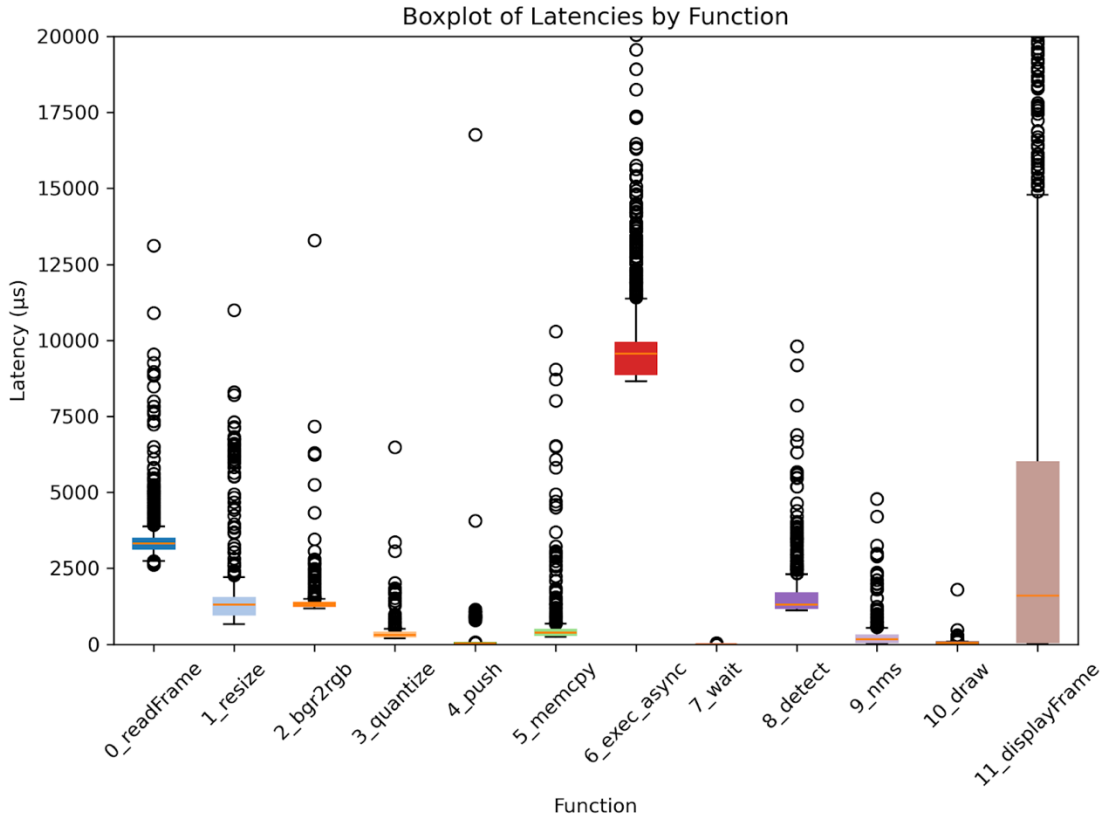
- B4096 1コアとB1024 2コアでの並列処理で検証



1 Core vs. 2 Core

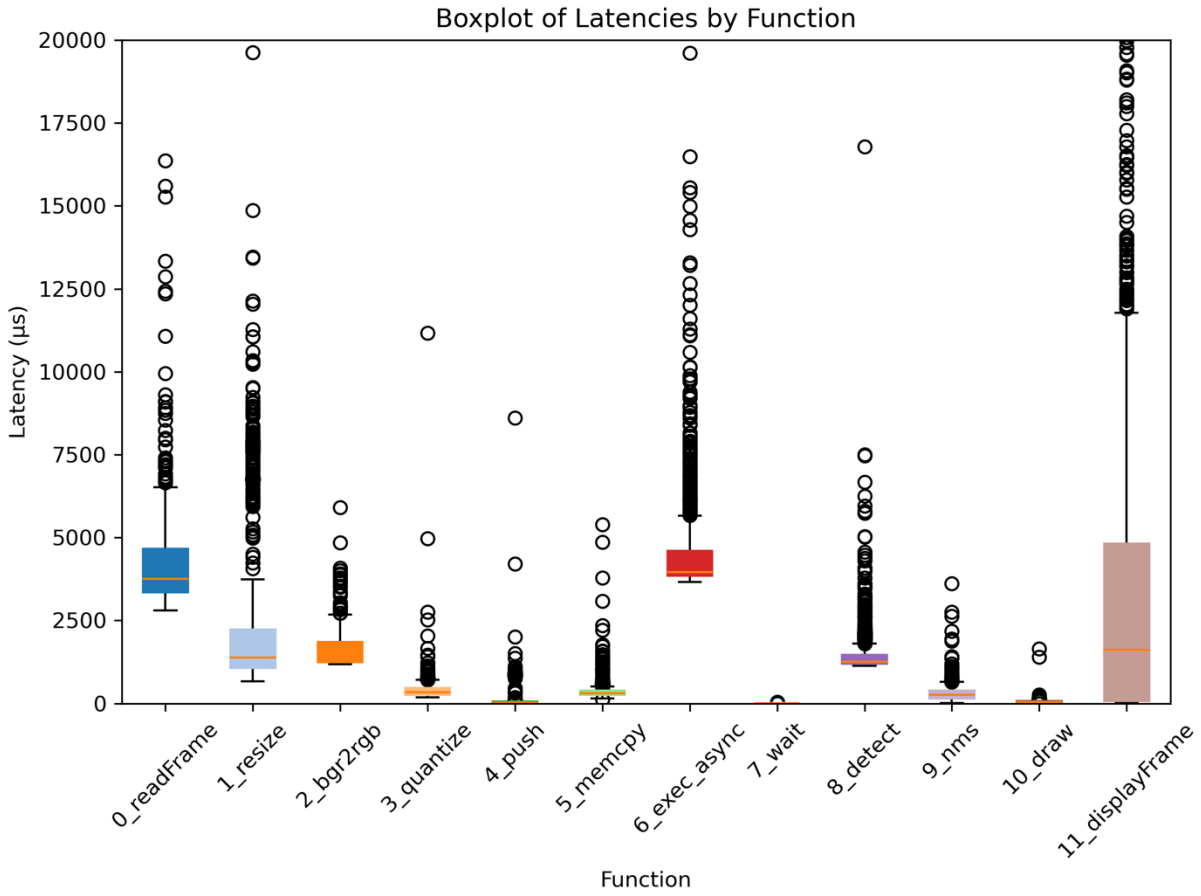
- ちょっと早くなった

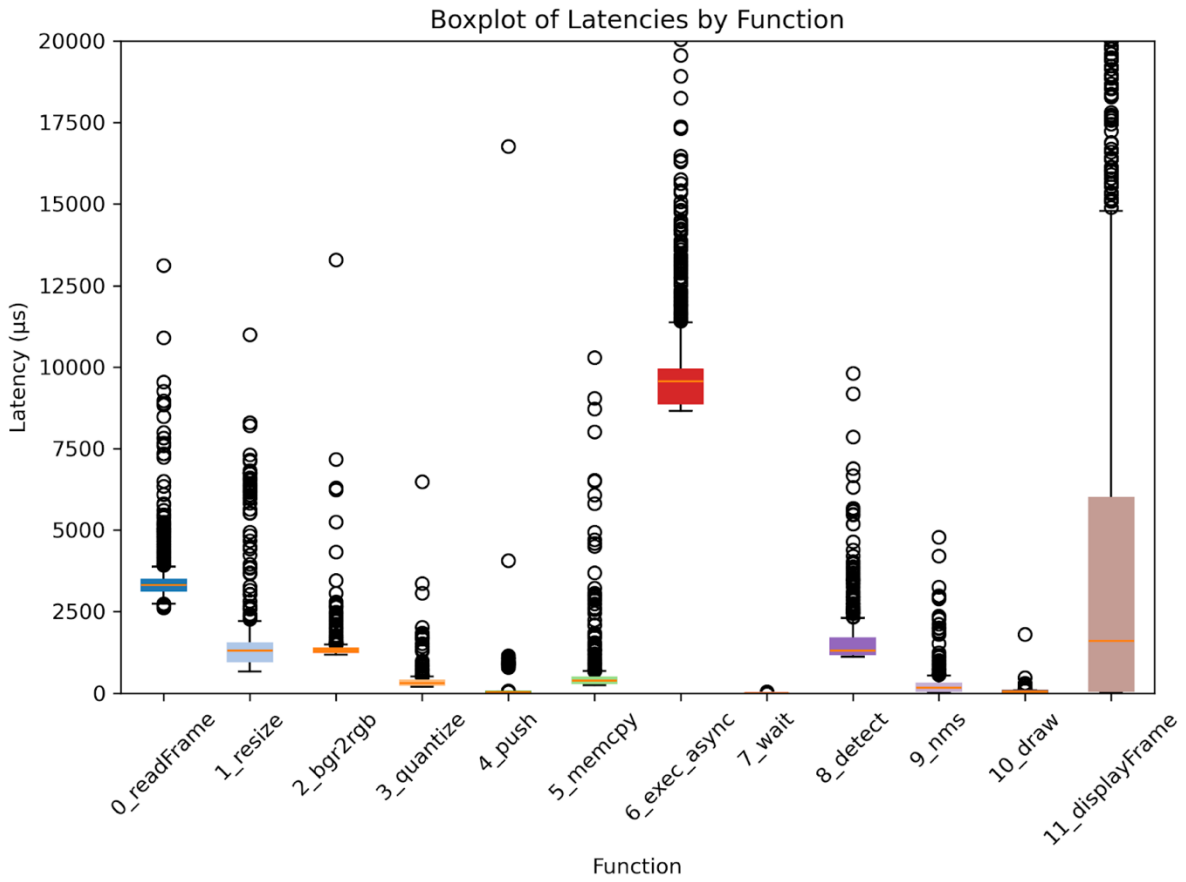




1 Core (300Hz) vs. 2 Core

- いや、同じか





実際に動かしてみる

大会結果

- 我々は5位でした
- 当日の様子：

チーム番号	FPS	mAP	Score
13	341.6	22.04	7529
15	177.8	24.55	4365
10	49.9	86.47	4315
3	50.2	59.25	2974
14	52.3	43.61	2281
1	75.4	24.34	1835
9	49.2	36.07	1775
8	51.6	28.03	1446
16	50.7	20.67	1048
12	10.9	80.76	880.3
4	14.8	58.1	859.9
6	6.6	93.47	616.9
18	83	6.1	506.3
5	7.3	47.46	346.5
17	7.3	47.41	346.1
19	2.3	47.46	109.2
2	2.3	47.41	109
7	1.13	22.41	25.32
11	0	0	0

2位チームとの差分振り返り

- データセットの学習、スレッド数が5だった

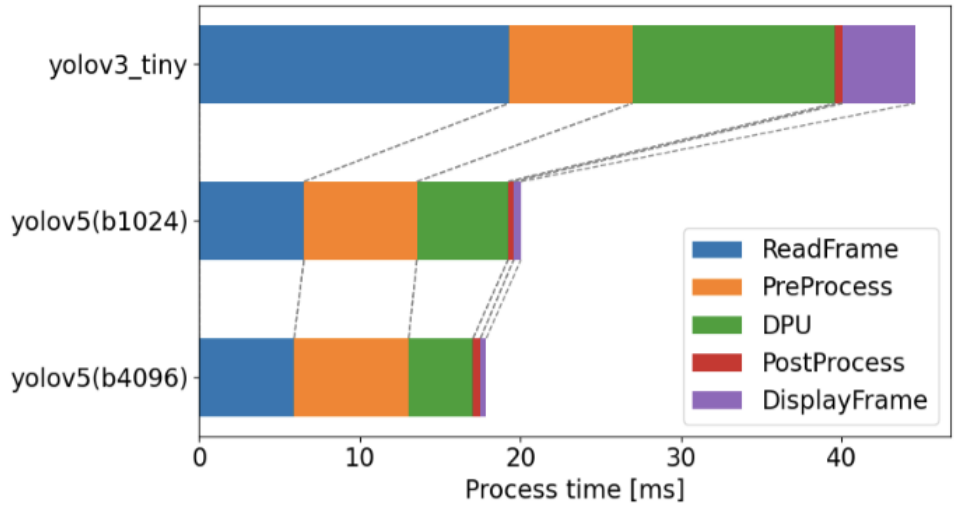


図 1　実装した YOLOv5n と元の YOLOv3-tiny の処理時間の比較

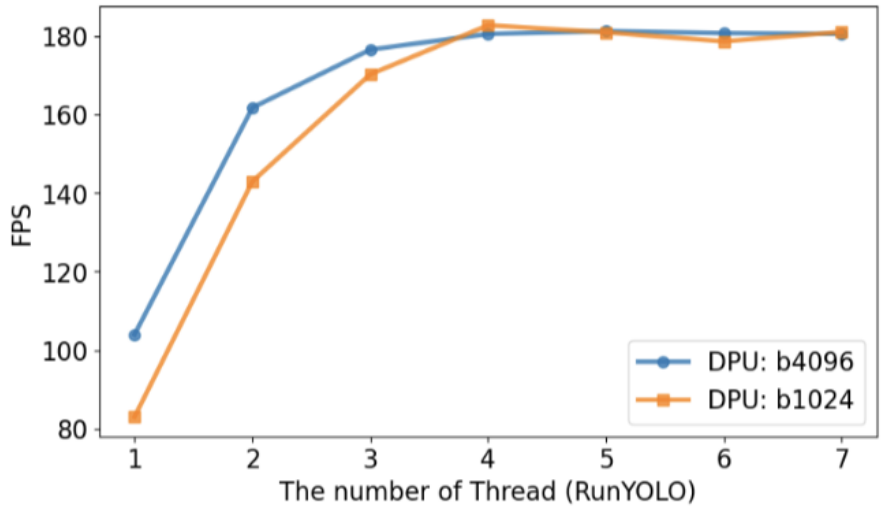


図 2　YOLOv5n における **RunYOLO** のスレッド数に対する FPS

まとめ

- FPGAハッカソンに参加したおかげで簡単にFPGAxAIが実現できた！
- 高精度かつ高速な推論をKV260を用いることに行うことができた
  - Vitis-AIとDPUを用いてFPGAによるYOLOの高速化を行った
  - ボード上のソフトウェアを改良することで、前処理・後処理の高速化を行なった

謝辞

- 素晴らしい機会を作って頂いたハッカソン実行担当の天野英晴先生に感謝致します。

ご清聴ありがとうございました！