Vol.2020-HPC-177 No.2
2020/12/21

Intel HD Graphics における Resnet50 の高速化

近藤鯛貴 [1,a)]　竹田大将 [1]　佐藤裕幸 [1]

概要：近年 AI ブームに伴い，小型ボード上で Deep Neural Network(DNN)の推論処理をリアルタイムで行うことの需要が高い．Intel HD Graphics は Intel 社 SoC に搭載された統合 GPU であり，同社の OpenVINO ツールキットを用いることで GPGPU による DNN 推論が可能である．しかし，GPGPU 実装は性能を最大に引き出すことが難しい OpenCL を用いており，また実装アルゴリズムも最適であると言い難い．本研究ではアセンブリ並みのパフォーマンスを引き出すことのできる C for Metal 言語を用いて Resnet50 を実装，さらに行列積の最適化や Winograd 実装を行った．結果として OpenVINO と比較し，畳み込み層のカーネル 3x3 は Winograd 実装により最大 37.28%，1x1 は行列積実装および最適化により最大 23.95%の高速化となり，推論時間は最大 23.53%の高速化に成功した．

キーワード： Intel HD Graphics，Deep Neural Network，GPGPU，SIMD

1. はじめに

近年，遠隔サーバなしに手元のコンピュータで計算処理を完結するエッジコンピューティングの注目が高い．特に Deep Neural Network(DNN)の社会実装において必要とされており，通信インフラに依存しないことから安定したスループット，通信コストの削減，セキュアなサービス運用等が期待される．エッジコンピューティングに用いられるコンピュータは多くの制約により，小型で低消費電力・高性能な物が望まれる．近年のエッジ需要の増大に伴い各半導体企業がこの要件を満たすコンピュータを数多く展開しており，特に DNN の計算に適した構造を持つことから高性能な GPU を搭載する小型計算機が多い. 小型計算機向けの GPU は SoC(System on a Chicp)に内蔵された Integrated GPU が主流である．高性能 GPU を持つ代表的な SoC として NVIDIA 社の Tegra シリーズと Intel 社の SoC が挙げられる．TegraGPU は同社の Discrete GPU と同様のアーキテクチャ構造を持つことから CUDA をはじめとする既存の GPGPU 環境が動作するため, 対応ソフトウェアも多い. 対して Intel 社の SoC 内蔵 GPU である Intel HD Graphics(iHD)は高いグラフィック性能を必要としない低消費電力向け GPU として設計されていることから Discrete GPU と比較して低性能であり，絶対的な性能を必要とする高性能計算では用いられず，NVIDIA 社の GPU に比べて GPGPU の活用事例は少ない．しかし，iHD は小型計算機に限らず，2010 年から今日までの Intel 社の SoC に幅広く搭載されており，非常に普及している．普及した計算機で高速な DNN 推論処理を実現することは, DNN の社会実装に強く貢献するといえる.

iHD の DNN ソフトウェアとして OpenVINO がある．OpenVINO は Intel 社の CPU・GPU・FPGA 等でコンピュータ・ビジョン, DNN の開発を支援するツールキットであり，これを用いることによって iHD による DNN 推論を行うことが出来る．iHD の DNN 開発は元は clDNN と呼ばれるプロジェクトで行われており，OpenCL を用いて実装されている. 現在は OpenVINO に統合され開発は継続されている．iHD の GPGPU 環境として OpenCL の他に C for Metal(CM)がある. CM は 2018 年に公開された C 言語ベースの GPGPU プログラミング言語であり，Intel 社によると iHD にてアセンブリ並みのパフォーマンスを発揮できるプログラミング言語であると主張されている[1]. このことから iHD の DNN 推論は C for Metal を用いた実装によりさらに高速になる可能性がある．また，DNN の代表的なモデルである Resnet50 を OpenVINO で動作させたところ，アルゴリズム面でも改良の余地があることが判明した．そこで，本研究では CM を用いて ResNet50 の推論処理を実装, さらにアルゴリズムの最適化を行い高速化を図った．具体的には CM を用いた Winograd アルゴリズムの実装と iHD のアーキテクチャ・製品特徴を考慮した行列積の最適化を行った.

本稿の構成は 2 章で Intel HD Graphics について説明し, 3 章では ResNet50，畳み込み層の計算手法について説明する．4 章，5 章では，Winograd と行列積の実装手法について述べる．6 章では本研究の実装と OpenVINO を比較し評価を行う．最後に 7 章をまとめとする.

2. Intel HD Graphics

本章では Intel HD Graphics のアーキテクチャ仕様と GPGPU 環境について述べる．詳細なアーキテクチャ構造と本研究で用いる C for Metal 言語の仕様は[2]を参照.

2.1. アーキテクチャ

Slice 単位の構成であり，1 つのスライスに SubSlice と L3 Data Cache がある．SubSlice にはコアとなる Execution Unit(EU)と Sampler と呼ばれる L1,L2Cache が搭載されている．EU はロックステップで動作する 4way-SIMD FPU x 2 個と SMT による 7 個のハードウェアスレッドを搭載している．スレッドごとにレジスタファイルを有しており，8x32bit のベクトルレジスタを 128 個，7 スレッドあることから 28KB/EU のデータを保持できる．SIMD の各レーンは命令を実行するか否かを決める Predicate レジスタがあり

1 岩手県立大学大学院ソフトウェア情報学研究科
a) g231s009@s.iwate-pu.ac.jp

SIMT 実行可能である．Slice,SubSlice,EU の数は製品モデルやアーキテクチャの世代によって変わる

2.2. GPGPU 環境

iHD の GPGPU 環境は大きく OpenCL,OpenCL SubGroups 拡張，C for Metal の３種類に分けられる．

(1)OpenCL

SIMT プログラミングモデル．OpenCL は他計算資源と同様のコードで動作するよう汎用に作られている．iHD では OpenCL は性能を発揮することが難しく，Intel 社公式の記事では行列積が，理論ピーク性能に対して数%程度の性能率にとどまっている [3].

(2)OpenCL SubGroups 拡張

SIMD を意識する SIMT プログラミングモデル．OpenCL に SIMD として動作するスレッド群である SubGroups を定義して iHD 独自の関数を追加したプログラミング言語である．SubGroups スレッド数は EU の SIMD 幅に依存しており，8,16,32 から選択可能である．OpenVINO や iHD 用の BLAS で用いられており, OpenCL より高いパフォーマンスを発揮することが出来る．

(3)C for Metal

SIMT 機構を活用した SIMD プログラミングモデル. 2018 年に公開された Intel GPU 専用 GPGPU プログラミング言語である．アセンブリに近いパフォーマンスを発揮できると Intel 社公式の記事では説明されており，ハードウェアを意識した実装が可能となる．カーネルコードは EU のレジスタ量である 4KB 以内に収める必要がある. リファレンスでは SIMD プログラミングモデルであると主張しているが，AVX や ARM NEON 等の SIMD とは仕様に相違点があり，特徴として命令ごとに SIMD 幅，Stride を指定可能であることが挙げられる．これは iHD が持つ SIMT 実行に必要な機構を SIMD プログラミングに活用したものであると推測される．

iHD は NVIDIA 社や AMD 社の GPU に比べてレジスタ量が非常に少ない．SIMT モデルは変数（レジスタ）のスコープから物理的には共有可能なレジスタでも SIMT スレッド間で共有することが難しい. 例えば CUDA ではスレッド間のデータ共有に共有メモリかシャッフル命令を用いる必要がありシームレスではない．SIMD モデルの利点はスレッドが SIMD を扱うことからこのような制約はなく，最大までレジスタを扱うことが出来る．また，C for Metal においては上述の SIMD 操作から，レジスタ操作も柔軟に行うことが可能で，レジスタの境界や SIMD レーンを意識することなくプログラミングが可能である．このプログラミング特性は後述する畳み込み層の計算手法である Winograd アルゴリズムの計算に適している．

3. ResNet50

Residual Networks(ResNet)は 2015 年の ImageNet Large Scale Visual Recognition Competition(ILSVRC)で優勝した DNN モデルである．エッジ向けデバイスや DNN フレームワークの性能評価に多く用いられており, DNN の実運用において重要視されている．

ResNet50 は 57 層からなるネットワークであり，その内畳み込み層が 53 層を占めている. 畳み込み層とは名前の通り畳み込み演算を行う層で，ResNet50 をはじめとする Convolution Neural Network(CNN)では頻出する計算アルゴリズムである．よって Resnet50 の高速化とは畳み込み層の高速化ともいえる．畳み込み層の計算方法は通常の畳み込み演算の他に im2col と Winograd アルゴリズムがある．本稿では畳み込み層の入力データのチャンネル数を in_ch, 縦幅と横幅をそれぞれ in_h,in_w, 出力データのチャンネル数を out_ch,縦幅と横幅を out_h,out_w と表記する．

3.1. im2col

im2col とは畳み込み計算が積和演算であることに着目し，入力データを行列に変換した後に行列積として計算する手法である． 1 ピクセル計算するためにカーネル 3x3 であれば in_ch x 3 x 3 の積和を行う．ここで 3x3 の畳み込みに用いるデータは他のピクセルと重複することから行列積の入力データ量は 9 倍に増えてしまうが，行列積として計算することによりキャッシュヒット率が向上しデータ転送を許容できるメモリ帯域であれば結果的に高速計算が可能となる．カーネルサイズが 1x1 では入力に変更なく行列積として計算できる．

3.2. Winograd

Winograd は畳み込み計算を行列の要素ごとの乗算に変換・計算し，逆変換で出力を得る手法である[4]．Winograd F(3x3,2x2)を例とする．F(3x3,2x2)は畳み込みカーネル 3x3 で出力 2x2 を計算するという意味を持つ． 1 回の計算で出力 2x2 を得ることから入力を重複した 4x4 に切り出し，変換・乗算・逆変換を行う．そのためデータ量は(out_h/2) x (out_w/2) x 4 x 4 に増加する．入力データの変換は 2 つの変換行列との行列積により得るが，変換行列は 1 と 0 で構成されており，ベクトルの加算と減算で行う．カーネルも変換する必要があるが，推論処理においてはあらかじめカーネルデータが決まっていることから事前に変換を行うことが可能である. 主計算である行列の要素ごとの乗算は in_ch 回の積和演算であり，独立した複数の行列積として計算できる. カーネル 3x3 は 16 個の独立した行列積となる．データ量が増え，計算量が減少する傾向のアルゴリズムであるため，通常の畳み込みと比べて高速になるかは対象とする計算機の演算性能とメモリ帯域，計算サイズを考慮し，選択する必要がある．本研究ではすべてのカーネル 3x3 にて Winograd を適用した．Winograd の計算式を式(1)に示す．

ここでdは切り出した入力データ, gは畳み込みカーネル, GとBは変換行列，Aは逆変換の行列である．⦿は行列の要素ごとの乗算である．各変換行列は計算する畳み込みカーネルのサイズによって変化する．

$$output = A^T\big((GgG^T)\odot(B^TdB)\big)A \quad (1)$$

3.3. Winograd Stride2

Winograd Stride2は，2ピクセル間隔で計算する畳み込み層を Winograd 化したアルゴリズムである[5]．Winograd Stride1 と同じように通常の畳み込み演算と比べ計算量が減り，データ量が増える傾向にあり，主計算を行列積に変換可能であることからメモリ帯域が強いコンピュータで高速に計算することが可能である．文献[5]ではNVIDIA社のGPUで通常の畳み込み計算と比べカーネル3x3で1.44倍，5x5で2.04倍，7x7で2.42倍の高速化に成功している．Resnet50の推論ではカーネル7x7と，使わない無駄なデータを計算対象外とするStrideの持ち上げ最適化により発生するカーネル3x3のStride2がある． 7x7 stride2は後述の理由によりiHDでは通常の畳み込みより理論的に高速にならないことが判明したため割愛する．

3x3 Stride2のアルゴリズムについて述べる．処理概要図を図1に示す．

(1) 入力データを出力2x2を計算する5x5に切り出す．
(2) 入力5x5とカーネル3x3を図1の番号ごとにグループ分けする．
(3) 各グループごとに畳み込み計算をする．
(4) 各グループの畳み込み出力を加算し，2x2のoutputを得る

ここで(3)の畳み込み計算はデータの重複が発生しないためWinogradで計算するほうが効率的である．

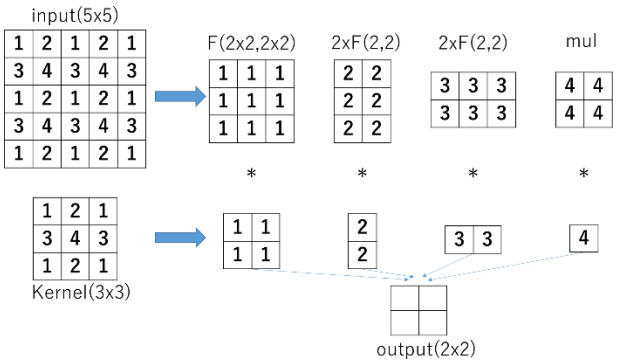

図 1：Winograd 3x3 Stride2

4. Winograd 実装

4.1. 概要

Winogradの CM 実装は２パターンある．一つは１スレッド内のレジスタを活用し, 変換･計算･逆変換を１つの GPU カーネルで完結する手法である. 計算途中に DRAM と通信しないことからデータ量増加を防ぎ，計算量を削減することができる．もう一つは行列積に変換する手法である．Winograd の行列要素ごとの乗算を行列積に変換する．入力データの変換で得る行列群を要素ごとの行列に配置するため，一度変換後データを DRAM に書き戻す必要がある．また，変換・逆変換処理と行列積ではスレッド粒度も異なるためそれぞれ別の GPU カーネルとして実装する必要がある．理論的にはデータ量の増大を防ぎ，計算量も削減できる前者の実装が効果的にみえるが，データの重複転送とキャッシュの局所性を確保することが難しく，結果的に行列積実装が高速となった．本稿では変換処理を Pre Process，計算処理を Convolution Process，逆変換処理を Post Process と表記する．

4.2. 3x3

カーネル 3x3 は 16 個の行列積に変換する．Pre Process と Post Proces の変換行列との積は行・列ごとの加算減算とすることが可能である．CM ではデータを２次元ブロックで扱うことを推奨しており，行列に対して行・列ごとに SIMD 演算できる．iHD は 1EU あたり 8way-SIMD をもつため， 1 スレッドが 1 枚の行列変換の担当では SIMD 演算器を活用できない．1 スレッドの粒度はレジスタに収まる最大量を計算し，SIMD 演算器の活用と入力データの重複転送の削減を行った．Convolution Process で行う複数個の行列積はスレッドで割り当てており，実装に関しては 5 章の行列積最適化で述べる．

4.3. 3x3 stride2

3x3 Stride2 において各グループで計算する Winograd は同じ変換行列を用いることから実際の計算では図１のように行列をグループ分けする必要はない．それぞれ行や列ごとに同様の計算をするため SIMD でまとめて行うことが可能である．Convolution Process では， 25 個の行列積を計算する．畳み込みカーネルからなる行列はグループ２，グループ３,グループ４内では重複しているため 16 個の行列とすることが可能である．

4.4. 7x7 stride2

7x7 も 3x3 と同様な行列変換処理で 81 個の行列積として計算することが可能である．Resnet50 の 7x7 stride2 では入力が 3x230x230，出力が 64x112x112 であり，Winograd に変換すると行列積は入力が 3,136 x 3 ,カーネルが 3 x 64 がそれぞれ 81 個となる. これらの情報と本研究の評価にも使われている Intel SoC である intel i3-8109U を用いてプログラムの演算性能を考察する. intel i3-8109U は 38.4GB/s のメモリ帯域と理論性能 806.4GFLOP/s の GPU を有している．ルーフラインモデル[6]における達成可能な GFlop/s(2)を用いて算出すると本プログラムにおける演算性能は約 54.971GFLOP/s であることがわかる．intel i3-8109U の GPU は理論性能 806.4GFLOP/s であることから演算性能は理論性能の 6.8%程度であり，高い性能率を出すことは不可能である．

7x7 Stride2 Winograd が intel i3-8109U においてデータ量

増加の問題から高い性能率を出すことが不可能である結論となった．計算量・データ量が違う通常の畳み込みとWinograd のどちらを採用するかについて理論値から考察する．各アルゴリズムのデータ量と計算量，演算性能から算出した理論計算時間を表 1 にまとめる．ここで Direct は通常の畳み込み計算を示す．計算量は 2.4 倍 Direct が多いがデータ量は Winograd の 5.6%程度であり結果として Direct が 6.06 倍高速に計算可能であるという結果になった．GPU はマルチコアな計算機であり，スレッド粒度やレジスタ量の関係上重複したデータ読み出しが発生するため実際の計算時間は理論値から乖離してしまうが，この SoC で Direct 実装を行ったところ 1.07msec で計算出来たことから Winograd の理論計算時間 1.652msec より高速であり Direct 実装を採用することが適切であるという結果となった．

$$\min(\text{Peak FLOP/s}, \text{Memory Bandwidth x (F/B)}) \quad (2)$$

表 1：各プログラムの理論性能

	計算量 (MFLOP)	データ量 (MB)	演算強度 (F/B)
Winograd (Convol)	93.023	64.981	1.431
Direct	225.093	3.703	60.835

	演算性能 (GFLOPS)	理論計算時間 (msec)
Winograd (Convol)	54.971	1.692
Direct	806.400	0.279

5. 行列積最適化

Resnet50 実装において畳み込み 7x7 Stride2 を除くすべての畳み込み層が行列積計算となった．行列積の実装は Relu 層の fuse や Winograd の複数行列同時計算等，層によって詳細な仕様は変わるが，行列積自体の実装法はすべて同様の手法で行っている．実装は CM のサンプルの Sgemm を参考にしており，詳細な実装法は [2]の行列積の項を参照．

本研究は iHD における高速な DNN 実装を目的としているが, iHD は多くの製品種類があり, 性能も大きく異なる．主に EU の数に種類があり，6EU~64EU(Intel Xe Graphics を含めると現時点で最大 96EU)ある．これに伴い，ハードウェアスレッド数や B/F 値も変わるため，製品によって最適な行列積実装が変わる．また，畳み込み層の計算サイズも様々な形, 特徴があるため, 層ごとに最適な実装が変わる．例として実際に計算するパターンで 49 x 2,048 と 2,048 x 512 の行列積では[2]の実装と同様に 1 スレッドのレジスタ数最大まで利用するように 1 スレッドの計算粒度を出力 8x64 と設定した場合，スレッド数は 56 スレッドとなる (ceil(49/8)*(512/64))．iHD が 6EU のときはハードウェアスレッドをすべて利用することが出来るが，64EU の場合は物理コアでさえも活用できない．そのため iHD と計算サイズによって GPU カーネルを切り替える必要がある．

もう一つの考慮対象として CM の仕様がある．CM のスレッド生成には X,Y,GroupX,GroupY を定義する．Group は CUDA の Block や OpenCL の Group と同様の意味を持っており Group が SubSlice，X,Y が SubSlice 内のスレッド数を定義するものだと推測される（X,Y の最大スレッド数が SubSlice 内のハードウェアスレッド数に依存しているため）．この X,Y の最適なスレッド設定の傾向が現時点で不明であり，総当たりで探索する必要がある．CM のサンプルにある行列積実装も同様に総当たりで探索している形跡がある．また，こちらも実験的に判明した事項であるが，スレッド X 方向が極端に少なく Y 方向が多い設定となったときに X と Y を反転させることで性能が改善したケースもあった．

これらを考慮し, iHD と計算サイズによって GPU カーネルを切り替えるシステムを作成した．iHD の行列積において出力行列サイズはスレッド当たり最大 512 要素分確保できることからそれ以内に収まる考え得るすべてのスレッド粒度の GPU カーネルを作成し， Resnet50 推論開始前に計算サイズごとに GPU カーネル x スレッド X,Y 設定数 x 2(スレッド反転)個の計算パターンから最も高速なものを総当たりで探索するようにした．

6. 評価

CM 言語による実装と OpenVINO で ResNet50 の推論時間を比較する．評価に用いたコンピュータとソフトウェアバージョンを表 2 にまとめる. 用いる GPU のクロック周波数は最大クロックを設定しているが, Celeron N4100 のみ低消費電力向けモデルであり，用いたコンピュータが BIOS でワット数を制限していたため最大クロックの維持が不可能であった．よって安定する 200Mhz に設定した．

表 2：評価環境

CPU	GPU[*1]			メモリ帯域 (GB/s)
	機種	EU@Mhz	GFLOPS	
Celeron N4100	UHD600	12@200	38.4	17.06
Celeron G4930	UHD610	12@1050	201.6	21.33
i3-9100	UHD630	24@1100	422.4	40.53
i5-6200U	HD520	24@1000	384.0	25.60
i3-8109U	Plus655	48@1050	806.4	38.40
i7-1065G7	PlusGEN11	64@1100	1126.4	42.66

CM	Linux_C_for_Metal_Development_Package_20190717

OpenVINO	OpenVINO_2021_1.110

評価方法について述べる．3x224x224 の画像を入力とする ResNet50 の推論処理を本実装と OpenVINO で計算時間を比較する．画像の入力から推論結果の取得までの 1,000 回実行の平均時間と畳み込み層のカーネルサイズごとの合計計算時間の比較を行う．また，行列積最適化の効果の検証として，すべての行列積が最大スレッド粒度で計算する実装（CM のサンプル実装）と最適化後の推論時間を各環境で比較する．OpenVINO の測定では推論時間は Inference Engine の Python API を用いて実装し，層ごとの計測は benchmark_tool のパフォーマンスカウンターを用いた．

推論時間を表 3 に示す. 推論時間は最大 23.53%高速化となったが，各 GPU でばらつきのある結果となった．OpenVINO は畳み込み層の実装は Winograd を用いておらず，我々の実装では 7x7 stride2 と 1x1 を除くすべての畳み込み計算が Winograd を採用している．Winograd はデータ量が増え計算量が減る傾向があるため，メモリ帯域が強いコンピュータで効果を発揮しやすい．これは結果にも反映されており，性能が低く，B/F 値が最も高い UHD600 で 23%の高速化となった．また，検証として，UHD610 をクロックが最低の 350MHz に設定(UHD610@350)し，B/F 値を向上させたところ最大クロック駆動より高速化率が向上した．よってメモリ帯域が強いコンピュータでは我々の実装が効果的であることが分かる．

次に畳み込み層の比較について述べる．UHD600, HD520, Plus655 でカーネルサイズごとの計算時間の合計値で比較する．EU 数の種類から PlusGEN11 も比較対象とすることが最善であるが，OpenVINO の benchmark_tool は一部の GPU で計算時間が安定せず，PlusGEN11 では Inference Engine を用いた推論時間より 1.5 倍高速に動作する．これは表 3 との結果の傾向から異常値であり，信憑性に欠けるため本稿では省いて考察する．比較結果を表 4 に示す．畳み込み 3x3 は評価 GPU すべてで我々の実装が高速になっており，Winograd は上述の通りメモリ帯域に効果が依存することから B/F 値が最も高い UHD600 で 37.28%の高速化に成功している．しかし，Winograd Stride2 は UHD600 でも低速となり，すべての GPU で効果は表れなかった．これは Stride2 は Stride1 と比べ，計算量の減少が少なく転送量がボトルネックとなったと考えられる．ResNet50 の主計算である畳み込み 3x3 と 1x1 はすべての GPU で高速になっているが，7x7 Stride2 は大きく低速になっている．本研究では上述の理由により 7x7 Stride2 の Winograd 化は行っておらず，計算アルゴリズムとしては OpenVINO と同様に通常の畳み込み計算で行っている．しかし，スレッド担当範囲や計算順序等の実装法が異なることから今後の課題として OpenVINO 実装の CM 移植が求められる．

次に表 5 の行列積最適化では実験に用いたすべての GPU で高速に動作しており，最大 10.56%の高速化に成功している．PlusGEN11 では最も高速化率が高いが，OpenVINO との推論時間比較では効果が表れていないことから Winograd が低速になっていると考えられる．

表 3：推論時間

GPU	計算時間(msec)		高速化率(%)
	本研究	OpenVINO	
UHD600	211.96	261.84	23.53
UHD610	47.43	51.63	8.86
UHD610@350	121.52	139.8	15.04
UHD630	24.70	26.08	5.58
HD520	29.67	31.68	6.77
Plus655	17.88	16.74	-6.38
PlusGEN11	15.24	14.73	-3.35

表 4：畳み込み層の比較

計算時間(msec)		7x7 Stride2	3x3	3x3 Stride2	1x1
UHD600	本研究	17.76	78.43	7.24	116.85
	OpenVINO	10.14	107.67	7.04	140.78
	高速化率(%)	-42.90	37.28	-2.75	20.48
HD520	本研究	2.01	11.62	1.20	14.28
	OpenVINO	1.19	14.24	0.95	17.70
	高速化率(%)	-40.79	22.54	-20.56	23.95
Plus655	本研究	1.07	6.11	0.72	8.12
	OpenVINO	0.56	6.68	0.52	9.02
	高速化率(%)	-47.66	9.35	-27.51	11.01

表 5：行列積最適化比較

GPU	最適化後	最適化前	高速化率(%)
UHD600	211.96	221.16	4.34
UHD610	47.43	50.83	7.17
UHD630	29.77	31.37	5.37
HD520	29.67	31.00	4.48
Plus655	17.88	19.13	6.99
PlusGEN11	15.24	16.85	10.56

7. まとめ

本研究では C for Metal 言語を用いた Intel HD Graphics の GPGPU による ResNet50 実装, さらに Winograd 実装と行列積の最適化を行った. 結果として Intel 社公式のソフトウェ

アである OpenVINO と比較し, 最大 23.53%の高速化に成功した．これは B/F 値が最も高いコンピュータでの結果であり, 本研究で実装した Winograd アルゴリズムはデータ量が増え，計算量が減る傾向から効果が顕著に表れたと考えられる．そのため，B/F 値が最も低いコンピュータでは効果が表れず，OpenVINO より 3.35%低速となった．行列積の最適化では最適化前に比べ 4.34%~10.56%高速になり，OpenVINO との比較では行列積で動作する畳み込み 1x1 は評価 GPU にて 11.01~23.95%の高速化となった．GPU や計算サイズによって OpenVINO の実装が高速となった層もあり, 今後の展望として OpenVINO の実装を CM に移植･GPU や計算サイズによって最適な計算アルゴリズムを選定するプログラムを作成することが求められる．また，Intel HD Graphics の後継として Intel Xe Graphics が登場し, エッジ向けの SoC でも上位モデルは Xe が採用されていることからこちらの対応も必要となる．アーキテクチャは iHD と類似しているため本研究の知見も活かすことが可能であると考えられる． C for Metal は現在も更新は盛んにおこなわれており，Intel Xe Graphics の対応も期待される．

謝辞

本研究は Idein 株式会社でのインターン・アルバイトにて実施された取り組みである．ここに深謝の意を表する．

参考文献

[1] Intel,” C for Metal Development Package | 01.org" , https://01.org/c-for-metal-development-package
[2] 近藤鯛貴, 竹田大将, 佐藤裕幸, “C for Metal 言語を用いた Intel HD Graphics の GPGPU 用途における性能評価”, 研究報告ハイパフォーマンスコンピューティング（HPC）, Vol.2019-HPC-172 No.15, pp1-6
[3] Intel,“SGEMM for Intel® Processor Graphics“, https://software.intel.com/content/www/us/en/develop/articles/sgemm-for-intel-processor-graphics
[4] Andrew Lavin, Scott Gray, “Fast Algorithms for Convolutional Neural Networks”, Proceedings of the IEEE conference on computer vision and pattern Recognition (CVPR), Jun. 2016, pp. 4013–4021.
[5] Juan Yepez, Seok-Bum Ko, “Stride 2 1-D, 2-D, and 3-D Winograd for Convolutional Neural Networks”, January 2020, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, PP(99):1-11, DOI I: 10.1109/TVLSI.2019.2961602
[6] Samuel W. Williams, Andrew Waterman, David A. Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures”, April 2009, Communications of the ACM 52(4):65-76, DOI: 10.1145/1498765.1498785

*1:
UHD:Intel UHD Graphics
HD:Intel HD Graphics
Plus:Iris Plus Graphics