Intel AVX-512 命令を用いた複数のモジュラ逆数計算の高速化

Fast Multiple Multiplicative Inverses Using Intel AVX-512 Instructions

高橋 大介 (Daisuke Takahashi)[1]
[1] 筑波大学計算科学研究センター (Center for Computational Sciences, University of Tsukuba)
e-mail : daisuke@cs.tsukuba.ac.jp

1 はじめに

モジュラ逆数は，計算数論や暗号などの分野で広く用いられている．モジュラ逆数は拡張 Euclid の互除法を用いて計算できるが，法が 2 のべき乗の場合は Newton 法が効率的であることが知られている [1, 2]．w を整数演算命令のビット幅とした場合，2^w を法をとする複数のモジュラ逆数計算を考える．このような計算は，複数の法に対する Montgomery 乗算や，複数の除数に対する exact division を計算するときに現れる．Intel Advanced Vector Extensions 512 (Intel AVX-512) は 512 ビットのベクトル命令セットである．本論文では，Intel AVX-512 命令を用いて 2^w を法とする複数のモジュラ逆数計算を高速化し，性能評価を行った結果について述べる．

2 モジュラ逆数

法 $N > 1$ に関する整数 a のモジュラ逆数とは，$ax \equiv 1 \pmod{N}$ となるような整数 x である．a の N を法とする逆数が存在するための必要十分条件は，a と N が互いに素（すなわち，最大公約数 $\gcd(a, N)$ が 1）となることである．

Algorithm 1 モジュラ逆数 $a^{-1} \bmod 2^{64}$ に対する Newton 法 [1]

Input: a such that $0 < a < 2^{64}$, $2 \nmid a$
Output: $x = a^{-1} \bmod 2^{64}$
1: $x \leftarrow \{(3a) \oplus 2\} \bmod 2^{64}$
2: **for** i **from** 1 **to** 4 **do**
3: $\quad x \leftarrow x(2 - ax) \bmod 2^{64}$
4: **return** x.

モジュラ逆数 $a^{-1} \bmod 2^{64}$ に対する Newton 法 [1] を Algorithm 1 に示す．ここで，$(3a) \oplus 2$ は modulo 2^5（5 ビット）の正確なモジュラ逆数であり [1]，$\oplus$ は排他的論理和演算を表す．Newton 法は 2 次収束するので，$a^{-1} \bmod 2^{64}$ を得るには 4 回の反復で十分である．

3 提案手法

モジュラ逆数 $a^{-1} \bmod 2^{52}$ に対する 2 次収束の Newton 法と 3 次収束の Newton 法の変形を組み合わせた提案手法を Algorithm 2 に示す．ここで，a は modulo 2^3（3 ビット）の正確なモジュラ逆数である [2]．$a^{-1} \bmod 2^{52}$ を得るには 3 次収束の Newton 法の変形を 2 回，2 次収束の Newton 法を 1 回用いれば十分である．Intel AVX-512IFMA（Integer Fused Multiply-Add）命令を用いて，Algorithm 2 の 52 ビット実装を行った．

性能評価にあたっては，複数のモジュラ逆数計算に対して，Intel 64 命令を用いた Algorithm 1 の 64 ビット実装，Intel AVX-512 命令を用いた Algorithm 1 の 64 ビット実装および Algorithm 2 の 52 ビット実装の性能の比較を行った．モジュラ逆数計算のバッチサイズを 512 から 4096 に変化さ

Algorithm 2 モジュラ逆数 $a^{-1} \bmod 2^{52}$ に対する 2 次収束の Newton 法と 3 次収束の Newton 法の変形を組み合わせた提案手法

Input: a such that $0 < a < 2^{52}$, $2 \nmid a$
Output: $x = a^{-1} \bmod 2^{52}$
1: $x \leftarrow a$
2: **for** i **from** 1 **to** 2 **do**
3: $\quad t \leftarrow (1 - ax) \bmod 2^{52}$
4: $\quad x \leftarrow \{x + x(t + t^2)\} \bmod 2^{52}$
5: $x \leftarrow x(2 - ax) \bmod 2^{52}$
6: **return** x.

表 1. 複数のモジュラ逆数計算の性能 (Invmod$\times 10^9$/s)

Batch size	Intel 64	Intel AVX-512	
	Algorithm 1 (64-bit)	Algorithm 1 (64-bit)	Algorithm 2 (52-bit)
512	0.38605	2.40270	4.60871
1024	0.38681	2.40460	4.60600
2048	0.38712	2.40560	4.62205
4096	0.38731	2.40201	3.95695

せた．入力する整数 a は，64 ビット実装では 1 から $2^{64} - 1$ の範囲の奇数の乱数であり，52 ビット実装では 1 から $2^{52} - 1$ の範囲の奇数の乱数である．各バッチサイズのモジュラ逆数計算を 100 万回実行し，その平均の経過時間から 1 秒あたりのモジュラ逆数計算回数（Invmod$\times 10^9$/s）を算出した．

評価環境として，Intel Xeon Platinum 8468 プロセッサの 1 コア，1 スレッドを用いた．コンパイラは Intel oneAPI DPC++/C++ Compiler 2024.0.2 を用い，コンパイルオプションは `icx -O3 -xSAPPHIRERAPIDS -qopenmp-simd -qopt-zmm-usage=high` を用いた．

Intel 64 命令および Intel AVX-512 命令を用いた複数のモジュラ逆数計算の性能を表 1 に示す．表 1 からバッチサイズが 512 の場合，Intel AVX-512 命令を用いた提案する 52 ビット実装は，Intel Xeon Platinum 8468 プロセッサにおいて Intel 64 命令および Intel AVX-512 命令を用いた 64 ビット実装と比較して，それぞれ約 11.91 倍および約 1.92 倍高速であることが分かる．

4 まとめ

本論文では，Intel AVX-512 命令を用いて 2^w を法とする複数のモジュラ逆数計算を高速化し，性能評価を行った結果について述べた．提案手法は 2 次収束の Newton 法と 3 次収束の Newton 法の変形の組み合わせに基づいている．性能評価の結果，提案手法が性能向上に有効であることを示した．

謝辞 本研究は，JSPS 科研費 22K12045 の支援によって行われた．

参考文献

[1] E. W. Mayer, “Efficient long division via Montgomery multiply,” *Comput. Res. Repos.*, 2016. [Online]. Available: https://arxiv.org/abs/1303.0328

[2] J. Hurchalla, “An improved integer modular multiplicative inverse (modulo 2^w),” *Comput. Res. Repos.*, 2022. [Online]. Available: https://arxiv.org/abs/2204.04342

複数ノードを用いた大規模な四倍精度行列乗算の実装

Towards a Multi-node Implementation of Large-scale Binary128 GEMM

河野 郁也 (Fumiya Kono)[1]
[1] 静岡理工科大学 (Shizuoka Institute of Science and Technology)
e-mail : kono.fumiya@sist.ac.jp

1 はじめに

高精度な数値計算が求められる科学技術分野においては，演算誤差の蓄積や収束性の問題により，浮動小数点数の精度が結果の信頼性に大きく影響する．特に Binary128 のような多倍長精度演算を必要とする問題は，数理最適化を含む数値線形代数の一部に見られる．しかし，長い語長の演算をハードウェアサポートする CPU は限られており，ソフトウェア実装では演算性能が非常に低下する．我々はこれまで，多倍長線形代数ライブラリ MPLAPACK[1] への組み込みを念頭に，GPU や FPGA 等のアクセラレータを活用し，Binary128 による四倍精度行列乗算の高速化を進めてきた [2].

GPU カーネルは，8×8 のブロック化を施した行列乗算を，Binary128 演算により実装している．一方，FPGA 向けの実装では，Binary128 の積和演算を行う 8×16 の演算要素 (PE) を正方形状に接続したシストリックアレイ構造を用いている．特に FPGA 実装では，行列サイズが大きくなるにつれて性能向上の傾向があり，Intel Agilex においては 24576×24576 の正方行列積に対して 91 GFlops を達成している．一方，GPU はボードメモリの帯域が高いことから，小規模行列においては FPGA よりも高い性能を示した．しかし，NVIDIA RTX 4090 および AMD Radeon 7900XTX におけるベンチマークでは，2000×2000 以上の行列積の演算性能が，それぞれ 76 GFlops，57 GFlops で頭打ちとなったことから，大規模な正方行列積においては FPGA 実装に優位性があると見ている．

これまでの研究では，1 ノード内における四倍精度行列乗算の高速処理に取り組んできたが，その応用可能性を追求するため，HPL ベンチマークのような大規模行列に対する演算の高速化に着目した．行列サイズが巨大化することで，1 ノード内のメモリにデータを保持できなくなるため，MPI を用いた複数ノードによる分散並列処理と，既存の四倍精度行列乗算カーネルを組み合わせることで，より大規模な問題を扱えるように構成した．本稿では，この MPI ベースの分散実装の概要と，複数ノードによる並列化の効果を評価した結果について報告する．

2 MPI によるノード並列計算実装

大規模な行列積 $C = AB$ を扱う際，GPU や FPGA などのアクセラレータに搭載されたデバイスメモリの容量は，ホスト側メモリ以上に強い制約となる．この制約を回避するために，本実装では行列 A, B, C をそれぞれ行方向・列方向に b 分割し，各ブロックをアクセラレータのデバイスメモリに収まる単位で処理する方針へと変更した構成を用いている [3]．分割された行列ブロックに対して，アクセラレータ上で部分積の計算を繰り返し実行し，最終的に C の全体を構成する．この部分積の計算を複数ノード間で並列に実行することで，大規模な四倍精度行列積の高速化を図る．

複数ノードによる大規模行列の処理にあたっては，親プロセスが行列全体のデータ管理，ブロック分割，および子プロセスへのデータ分配と結果の収集を担当する．一方，子プロセスは受信した行列ブロックに対して四倍精度の部分積を計算し，その結果を親プロセスに送信する．ゆえに，各子プロ

セスがアクセラレータを用いて計算を行う構成となる．2 ノードによる並列実行時には，親プロセスを含めた 3 ノードでの計算となる．

データ転送は，親プロセスが計算対象となるブロック番号を子プロセスに通知した後，対応する行列 A, B, C のブロックを順に送信する方式を採用している．子プロセスが前のブロックの計算で使用した行列 A を再利用可能な場合には，そのデータを使い回すことで送信量の削減を図った．通信には非同期メッセージパッシングを用い，親子プロセス間で `MPI_TEST` による定期的な監視を行いつつ，送受信の進行状況を確認する．また，各通信には計算対象のブロック番号に基づくタグを与え，データの整合性を保ちつつ通信の衝突を防いでいる．

3　ノード並列の性能評価

ノード並列化による効果を確認するため，2 並列でのベンチマークを行った．FPGA については，Stratix 10 を搭載した筑波大学のスーパーコンピュータ Cygnus(2025 年 3 月に稼働終了) において，計 3 ノード (各ホストのメモリ容量は 192GB）を用いて評価した．一方，GPU については，NVIDIA H100 を 2 基搭載する計算機 1 ノード (ホストメモリ 512GB) において，3 プロセスを起動し，2 台の GPU で並列計算を行った．

図 1 に FPGA における分割数 $b = 2$ における計算, 図 2 に GPU における分割数 $b = 2, 32$ における計算のベンチマーク結果を示す．Cygnus の稼働終了に伴い，2 台の Stratix10 においては行列サイズは $n = 16000$ までの検証であるが，1 台での計算と比べて約 2 倍の性能向上となっており，MPI 実装によるノード並列の効果は確認できる．GPU においては，32 分割により 2 ノードで $n = 94208$ まで計算できているが，行列サイズ増加に伴う MPI 通信の不安定さの制御が性能向上の妨げとなっている．今後，分割した大規模行列に対するプロセス間通信の改良を進めていく．

図 1: FPGA における 2 ノード並列の効果

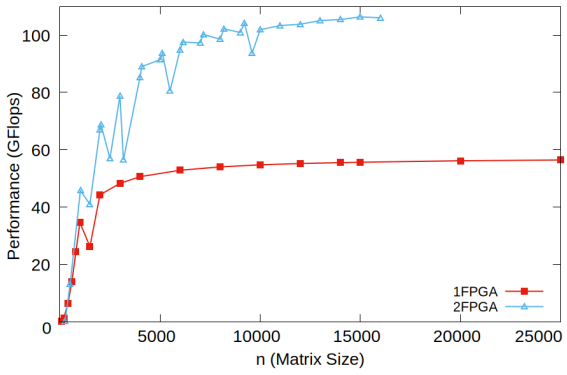
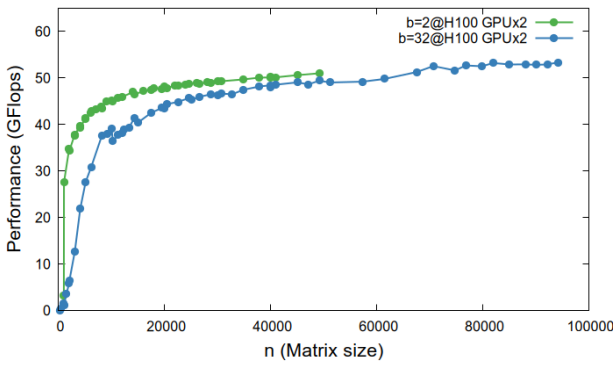


図 2: 分割数 b による GPU ノード並列計算の効果



参考文献

[1] Maho Nakata. (2022). MPLAPACK version 2.0.1 user manual.

[2] 中里直人, 河野郁也, 中田真秀： 様々な浮動小数点演算形式の評価プラットフォームとしての FPGA と GPU の比較，研究報告ハイパフォーマンスコンピューティング，Vol. 2025-HPC-198, No. 59, pp. 1–6 (2025).

[3] 河野郁也: Stratix10 FPGA による大規模な四倍精度行列乗算の性能評価, 日本応用数理学会 2024 年度年会

大学学部生へのHPC技術教育と機械学習・行列乗算の並列計算

Parallel Computing Practice in Machine Learning and Matrix Multiplication for Undergraduate HPC Education

河野 郁也 (Fumiya Kono)[1], 久保埜 成弥 (Seiya Kubono)[1], 打桐 大晃 (Hiroaki Utsugiri)[1]
[1] 静岡理工科大学 (Shizuoka Institute of Science and Technology)
e-mail : kono.fumiya@sist.ac.jp

1 はじめに

近年，様々な科学技術分野において，大規模なデータ処理や高精度な数値解析が求められており，その基盤として高性能計算 (HPC) 技術の重要性は一層高まっている．しかし，HPC 分野は幅広い数理や計算機科学の知識を要し，大学学部生を対象とした初学者向け教育は体系化が難しい．著者らの属する静岡理工科大学は，地方私立大学として多様な学生を受け入れており，学生の学力差も顕著であるため，学部授業で踏み込んだ計算機システムや並列計算の教育は行えていない．研究室配属に際しても，多くの学生は成果物が視覚的に華やかな研究テーマを好み，それらの基盤たるマルチコア CPU や GPU アクセラレータのバックグラウンドを知ること無く卒業していく．

著者らは 2023 年に HPC 技術を中心とする研究室を設立し，現代計算機技術に関心のある学部生を集めて HPC 分野の基盤技術教育を進めている．研究室では自動運転技術を 1 つのターゲットとし，機械学習の低レベル実装 (C 言語によるニューラルネットワーク構築) や，Tensor コアを活用した行列演算の高速化といった複数のアプローチを通じて，学生の HPC 分野への関与を進めてきた．本稿は，当研究室で推進してきた HPC 技術教育に基づく卒業研究の成果の一例をまとめて報告する．

2 研究室配属から卒業研究着手までの並列計算教育

静岡理工科大学では，3 年生が夏休みを前に研究室に配属される．卒業研究に向けて，当研究室の学生は後期セミナー科目を通じて，約半年で以下の並列計算の基礎知識と技能を身につけていく．

1) C 言語の確認，特にメモリ管理 (ポインタ)
2) コンピュータアーキテクチャ論と MIPS アセンブリプログラミング
3) キャッシュ効果を意識した逐次計算の最適化法
4) CPU における並列化 (OpenMP によるスレッド並列，AVX 命令による SIMD 化)
5) GPU における並列化 (OpenACC，CUDA によるプログラミング)

当研究室の学生は，一定の数理やプログラミング能力を有する場合が多いが，開講科目だけでは HPC への入門には不足するのが実情である．メモリ動的確保等のポインタ操作の復習に始まり，英語資料を基にした計算機システムの輪講を通して，学生が自ら調べて知識を整理する期間を設ける．またアセンブラ演習を並行し，レジスタレベルでの CPU 上の命令実行の流れも理解させていく．

その後 C 言語プログラミングに戻り，行列積のブロッキングなどを題材に，メモリのアクセス順序とキャッシュ利用が演算性能に与える影響を調べ，逐次計算の効果的な実装法を学ぶ．そしてようやく並列計算の概念に触れ，マルチコアやベクトルレジスタといった現代的な CPU の構造を活かした計算法を実践する．最後に，GPU の階層的アーキテクチャやスレッドモデルを意識した GPU プログラミングに触れ，デバイス内の高速なメモリの利用を意識した実装法までを全員で習得する．

3　機械学習と行列乗算の並列計算

卒業研究として，機械学習分野における数値計算の高速化を目指した事例を示す．我々は，新しいハードウェアを用いた性能評価に積極的に取り組んでおり，学生は研究を通じて，新しいアーキテクチャによる計算機性能の差異を理解する．ここでは，Intel Core-i9 14900K をホスト CPU として，NVIDIA RTX4090 GPU を搭載した計算機を実験に使用した．

1 例目は，MNIST を題材としたニューラルネットワークの C 言語実装と高速化である．AI 計算は Python 実装が主流であるが，インタプリタ型言語のプログラムは低速な傾向にある．本研究では，リアルタイム物体認識のような認識精度と演算速度の両立が求められる問題への先駆けとして，手書き文字データセットである MNIST を対象に，C 言語ベースでニューラルネットワーク構築から GPU 高速化までを行い，Keras による Python 実装と性能を比較した．

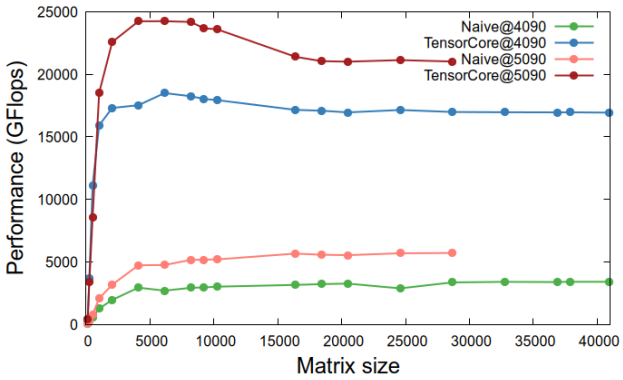
まず，入力層・隠れ層・出力層の 3 層ネットワークを，単精度演算による逐次計算で実装した．計算全体で約 98% を行列積が占めることを `gprof` により確認し，行列積部分を OpenMP や CUDA を用いて高速化した．各実装法における学習時間を表 1 に記す．一般的な Python 実装に比べ，C による独自の実装・並列化は実装コストが高いものの，より高速な学習処理が行えることを確認した．

2 例目は，Tensor コアを用いた行列のブロック化乗算である．機械学習の計算には行列積が頻出することから，近年の NVIDIA GPU には，半精度の小規模行列積を高速実行する Tensor コアが搭載されている．Tensor コアの性能検証は椋木ら [1] の前例があるが，本研究では RTX40, 50 シリーズの新しい世代の GPU による評価を行った．半精度の n 次正方行列の積 $C = AB$ に対して，Tensor コアに載せられる 16×16 で行列をブロック分割し，全てのブロックの積を Tensor コアを繰り返し用いて計算するプログラムを実装した．ブロック化した行列は WMMA API を通して専用のデータ型で保持し，Tensor コアで積和算を行う．図 1 は，RTX4090(及び 5090) での Tensor コアによる行列積の性能を，単精度でのブロック化行列積と比較したものである．最新世代のコンシューマ GPU では，Tensor コアを明に利用できる計算は，従来の 5 倍以上の性能向上が可能であると確認できる．

表 1: MNIST の各言語・並列化実装における学習処理の所要時間 (秒)

実装法	所要時間
Python (Keras 逐次)	1792
C (逐次)	34.61
CuPy (GPU 並列)	5.87
C/OpenMP (24 並列)	5.54
C/OpenMP (32 並列)	3.99
C/CUDA (GPU 並列)	1.07

図 1: Tensor コア使用有無による行列積の性能比較



参考文献

[1] Daichi Mukunoki, Katsuhisa Ozaki, et al. 2020. DGEMM Using Tensor Cores, and Its Accurate and Reproducible Versions. In High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings. Springer-Verlag, Berlin, Heidelberg, 230–248.

FPGA を用いた多様な演算精度による画像認識アルゴリズムの性能評価

Performance Evaluation of Image Recognition Algorithms with Various Arithmetic Precisions on FPGA

栗原 宏綺 (Hiroki Kurihara)[1], 河野 郁也 (Fumiya Kono)[1],
[1] 静岡理工科大学 (Shizuoka Institute of Science and Technology)
e-mail : 2421009.kh@sist.ac.jp

1 はじめに

近年, AI 研究の分野において, 計算精度の低精度化が顕著な傾向としてみられる. この低精度化は, 大規模化が進む AI 開発において, 計算速度とエネルギー効率の改善に寄与するものである. 特に, 数千億パラメータを有する大規模言語モデル登場により, 従来の 32 ビット浮動小数点数（FP32）では計算資源使用量や計算コストの増大が課題となる. そのため, 性能向上との両立を図るべく, 低精度コンピューティングの利用と研究が盛んにおこなわている. しかしながら, 低精度化は丸め誤差の増大や桁落ち, 表現可能な範囲の縮小を招き, 結果として学習モデルを不安定化させるリスクを内包する.

一方, 実社会においては自動運転技術が市場に出始めている. 自動運転システムには, 小規模で安全性を優先しつつ, 最大限の性能を発揮することが求められる. そこで, 本研究では, 比較的小規模な FPGA ボードである Kria™ KV260 を対象とし, 手書き数字データセット「MNIST」を 3 層の DNN（Deep Neural Network）で処理する回路を Verilog HDL で設計・実装した. その上で EPOCH AI 社の調査 [1] を参考に, 近年 AI 分野で採用が進む複数の浮動小数点フォーマット（FP32, FP16, bfloat16, FP8）について, ハードウェア資源使用量と性能の比較を行い, その結果を報告する.

2 実験手法

本実験は, 自動運転技術をテーマとした ICCE の Design Contest[2] の使用機材, 及び弊研究室の昨年度卒業生による卒業論文を参考に以下の前提条件で開発を行なった. そのため, 実験は以下の条件で行うこととした.

- データセット: 正規化済み mnist データを使用する
- 数値形式: 固定小数点形式ではなく浮動小数形式を使用
- ターゲットデバイス:Kria™ KV260 を使用

データ処理に用いる DNN は, 入力層 784 ノード, 中間層 50 ノード, 出力層 10 ノードから成る 3 層構成である. 本実装にあたり, 主要な演算処理である浮動小数点数の乗算および加算を行うための演算回路を個別に設計した.

本研究で設計したプログラマブルロジック（PL）側の処理フローを以下に示す. 一連の処理には, パイプライン技術を適用し, スループットの向上を図った. なお, 予稿執筆時点では加算, 乗算回路のみの実装, 及び結果を掲載している.

1) 入力層から中間層への演算
2) 中間層から出力層への演算
3) 最終出力処理

3　評価

本稿執筆時点において, 乗算回路と加算回路の単体での論理合成を完了している. その結果, 基準となる FP32 形式からビット長を半減させた FP16 形式では, フリップフロップ（FF）およびルックアップテーブル（LUT）の使用量が約 50% 削減されることを確認した.

ただし, bfloat16 形式の実装においては, 現状でいくつかの課題が確認されており, これは今後の修正対象である. これらの初期結果は, ハードウェア資源の観点から, より集積度の高い大規模な AI 回路を実装できる可能性を示唆している.

一方で, 実験過程においては, 低精度化に伴う丸め誤差の問題が観測された. 特に FP8（E5M2）形式を用いた実験では, 本来 10 進数で「−4.5」として表現されるべき値が, 「−4.0」として表現されるなど、その低い表現能力ゆえに正しく処理されない事象が発生した.

また, 本設計では浮動小数点形式を採用しているため, 加算に 3 クロック, 乗算に 7 クロックを要している. これを固定小数点形式に置き換えることで, 演算遅延を短縮し, システム全体のスループットを向上できる可能性がある.

表 1. 演算精度別のハードウェア資源推定使用量の比較

小数点精度	FF(ADD)	LUT(ADD)	FF(MUL)	LUT(MUL)
FP32	311	349	117	75
FP16	165	168	54	37
bfloat16	151	135	78	118
FP8(E5M2)	80	71	46	31
FP8(E4M3)	88	68	46	33

4　おわりに

本研究では, DNN の推論処理における数値形式として浮動小数点数を採用した. 今後の展望として, これを固定小数点形式で再実装し, 処理クロック数やハードウェア資源消費量の観点から, その有効性を定量的に評価する.

さらに将来的には, 本カスタム回路による実装と, Kria KV260 にて使用することができる AMD 社製の AI 処理専用 IP「深層学習プロセッシングユニット（DPU）」を利用した場合との性能比較を行い, 特定用途向け回路設計の優位性について考察を進める予定である.

参考文献

[1] EPOCH AI Widespread adoption of new numeric formats took 3-4 years in past cycles, https://epoch.ai/data-insights/training-precision

[2] Kojima Akira, Design of Miniature Automatic Driving Vehicle Controlled by Camera Image Processing, 2024 IEEE International Conference on Consumer Electronics (ICCE), 2024, 1–2

