

CGLA 上の多様な AI ワークロードに向けた 統合ソフトウェアスタックの実装と今後の展望

安藤 拓翔[†] 宗片 吉史[†] 中島 康彦[†]

[†] 奈良先端科学技術大学院大学 先端科学技術研究科 〒630-0192 奈良県生駒市高山町 8916-5

E-mail: †ando.takuto.aq5@naist.ac.jp

あらまし AI ワークロードの多様化に伴い、推論基盤には電力効率を保ちながら異なるワークロードへ展開できる柔軟性が求められる。GPU は汎用性が高い一方で消費電力が大きく、専用 ASIC は高効率だが対象ワークロードが固定されやすい。FPGA は柔軟だが、動作周波数と実行性能に制約がある。そこで本稿では、これらの中間に位置する CGLA (CPU-Grounded Linear Array) 上で、`llama.cpp`, `whisper.cpp`, `stable-diffusion.cpp` などの ggml 系 OSS を対象に、共通する実装方法と初期評価を整理する。CPU 側の行列積・dot 積実装を IMAX へオフロードし、この実装方法を音声認識と画像生成にも適用した。既存結果より、大規模言語モデルのトークン生成フェーズは LOAD とホスト側処理に支配され、動的チャンク分割によりトークン解析フェーズは 1.62 倍に高速化した。Whisper 音声認識は Jetson AGX Orin 比 1.90 倍、RTX 4090 比 9.83 倍の電力効率を達成した。Stable Diffusion では FP16/FP32/整数型が混在したモデルの実装に成功した。これらを踏まえ、共通のフロントエンド、計測に基づくランタイム、方策に応じて切り替えるバックエンドから成るソフトウェアスタックの論点を示す。

キーワード CGLA, AI アクセラレータ, FPGA, ソフトウェアスタック, ランタイムシステム

Future Directions and Implementation of an Integrated Software Stack for Diverse AI Workloads on CGLA

Takuto ANDO[†], Yoshifumi MUNAKATA[†], and Yasuhiko NAKASHIMA[†]

[†] Nara Institute of Science and Technology 8916-5 Takayama, Ikoma, Nara, 630-0192 Japan

E-mail: †ando.takuto.aq5@naist.ac.jp

Abstract As AI workloads diversify, inference platforms must provide both power efficiency and the flexibility to support a range of workloads. GPUs are highly programmable but power-hungry, whereas dedicated ASICs are efficient but typically tailored to specific workloads. FPGAs are flexible, but their operating frequency and execution performance are often limited. In this paper, we present a common implementation approach and an initial evaluation for ggml-based OSS workloads on CGLA/IMAX, including `llama.cpp`, `whisper.cpp`, and `stable-diffusion.cpp`. In `llama.cpp`, we offload CPU-side matrix multiplication and dot-product kernels to IMAX and extend the same implementation boundary to ASR and image generation workloads. Our prior results show that LLM decode is dominated by LOAD and host-side processing, Q-Snap improves prefill performance by 1.62×, Whisper ASR achieves 1.90× and 9.83× higher energy efficiency than Jetson AGX Orin and RTX 4090, respectively, and Stable Diffusion is implemented with mixed FP16/FP32/quantized execution. Based on these observations, we discuss software-stack issues for CGLA, including a common frontend, a measurement-driven runtime, and policy-specialized backends.

Key words CGLA, AI accelerator, FPGA, software stack, runtime system

1. はじめに

AI アプリケーションは、LLM、音声認識、画像生成のように計算特性が大きく異なる一方で、推論時の消費電力と実行効率が共通課題になる。特にエッジ環境では、限られた電力、

帯域、メモリ容量の中で実用的な性能を実現する必要があり、サーバ環境でもモデルの大規模化に伴って消費電力とデータ移動コストの増大を無視できない。

GPGPU (General Purpose Graphics Processing Unit) は柔軟性が高く多様な AI アプリケーションを実行できるが、消費電力が大

きい。その一方で専用 ASIC (Application Specific Integrated Circuit) は消費電力が小さく高効率だが、一度設計すると別のアプリケーションへ適用しにくい。FPGA (Field-Programmable Gate Array) は柔軟だが、動作周波数と実行性能の制約が大きい。この間を埋める選択肢として、我々は再構成性とデータ移動制御を備えた CGLA (CPU-Grounded Linear Array) アーキテクチャ IMAX (In-Memory Accelerator eXtension) を開発してきた。

ただし、ハードウェアを用意するだけでは、複数アプリケーションにわたってソフトウェア資産は蓄積しない。既存の AI アプリケーションは OSS (Open Source Software) 上に実装されており、モデル実行、前後処理、量子化形式、実行制御がアプリケーションごとに異なる。そのため、CGLA を継続的な AI 実行基盤として用いるには、既存 OSS を大きく壊さずに取り込み、どこまでを共通実装として残し、どこからをアプリ依存として切り分けるかを整理する必要がある。

我々は、llama.cpp を出発点として、CPU 側の行列積・dot 積実装を IMAX にオフロードし、ホスト側の制御フローや KV キャッシュ管理は既存 OSS 側に残す構成を実装してきた [1]。さらに、この実装境界を whisper.cpp と stable-diffusion.cpp に広げ、複数アプリに共通する実装方法と初期評価を整理してきた。本稿で扱うのは、個々のアプリで最良性能だけを競うことではなく、OSS の一部修正で複数アプリへ接続できた実装方法と、そのときに現れる共通課題である。

本稿の目的は、AI アプリ実装を横断して、(1) IMAX へオフロードする共通実装をどこに置くべきか、(2) 初期評価から何が共通課題として見えるか、(3) それを継続的に扱うためにどのようなソフトウェアスタックが必要か、を整理することである。

本稿の主な貢献は次の三点である。

- AI アクセラレータ、GPU、FPGA、LLM アクセラレータ研究の流れの中で、IMAX/CGLA を柔軟性と効率の両立を狙う AI 実行基盤として位置づけた。
- llama.cpp, whisper.cpp, stable-diffusion.cpp を対象に、行列積・dot 積のオフロード、DMA、ホスト協調を共通する実装として整理し、既存結果に基づく初期評価をまとめた。
- これらの結果を踏まえ、データ転送律速、データ型混在、Python と C/C++ 実装の分断を共通課題として整理し、共通のフロントエンド、計測に基づくランタイム、方策に応じて切り替えるバックエンドから成り立つソフトウェアスタックの論点を示した。

2. 関連研究

2.1 AI アクセラレータ

AI アクセラレータ研究では、GPU を汎用実行基盤として用いつつ、より高い電力効率を狙う専用アーキテクチャが継続的に提案されてきた。Jouppi らの TPU は行列演算を中心に構成したデータセンター向け ASIC の代表例であり [2]、Chen らの Eyeriss はデータフローとメモリ階層を明示的に設計した DNN アクセラレータの代表例である [3]。また、Sze らは DNN ア

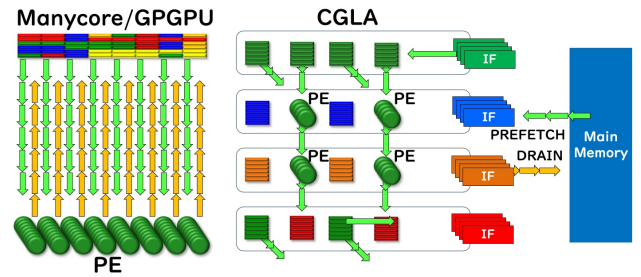


図1 Manycore系とCGLAアーキテクチャの違い。Manycore系では演算器とメモリの距離が相対的に速く、共有メモリ階層を介してデータを受け渡す。これに対しCGLAでは各PEの近傍にローカルメモリモジュール (local memory module: LMM) があり、隣接PE/LMM間でデータを直接受け渡ししながらパイプライン実行する。

クセラレータをデータ移動、データフロー、精度の観点から整理している [4]。さらに、MAERI は、多様な DNN 層やデータフローへの適応を狙う柔軟なアクセラレータとして位置づけられる [5]。Plasticine は、特定の DNN カーネルに閉じない coarse-grain 再構成型アクセラレータとして、より広い並列パターンを扱う立場を示している [6]。これに対し CGRA/CGLA 系アーキテクチャは、対象モデルごとの専用回路化ではなく、再構成性を保ったまま複数アプリケーションへ展開できる計算基盤を目指す立場にある [7]。IMAX アーキテクチャ上ではすでに SpGEMM, FFT, CNN, LLM などが実装されており、CGLA が単一用途ではなく汎用計算基盤として機能することが示されている [1], [8]。

2.2 LLM 推論ソフトウェアと量子化

ソフトウェア側では、LLM 推論の実行系と量子化の研究が進められている。PagedAttention/vLLM は、KV キャッシュ管理を含む実行方策がスループットと実装形態に強く影響することを示している [9]。また、SmoothQuant は、量子化形式の選択がカーネル構成と実行効率に強く関わることを示している [10]。

2.3 本研究の位置づけ

既報では、ggml 系 OSS を基盤として、ホスト CPU 側のテンソル管理や推論ループを保ったまま、行列積・dot 積の一部を IMAX にオフロードする構成が、LLM、音声認識、画像生成に適用されている [1], [11], [12]。既存研究では、量子化、メモリ管理、実行系最適化が個別に議論されているが [9], [10]、CGLA 実機上で複数アプリを横断して、どこまでを共通実装として再利用できるか、どのような計測形式と実行方策が必要かは十分に整理されていない。そこで本稿では、既存の実装資産を踏まえて、共通化できる OSS 修正とアプリ依存の部分として残る箇所を整理する。

3. CGLA/IMAX と具体的な実装と評価

3.1 IMAX3 の概要と共通する実装

図1に示すように、GPU のような Manycore 系では、多数の演算器が共有メモリ階層を介してデータを読み書きしながら並

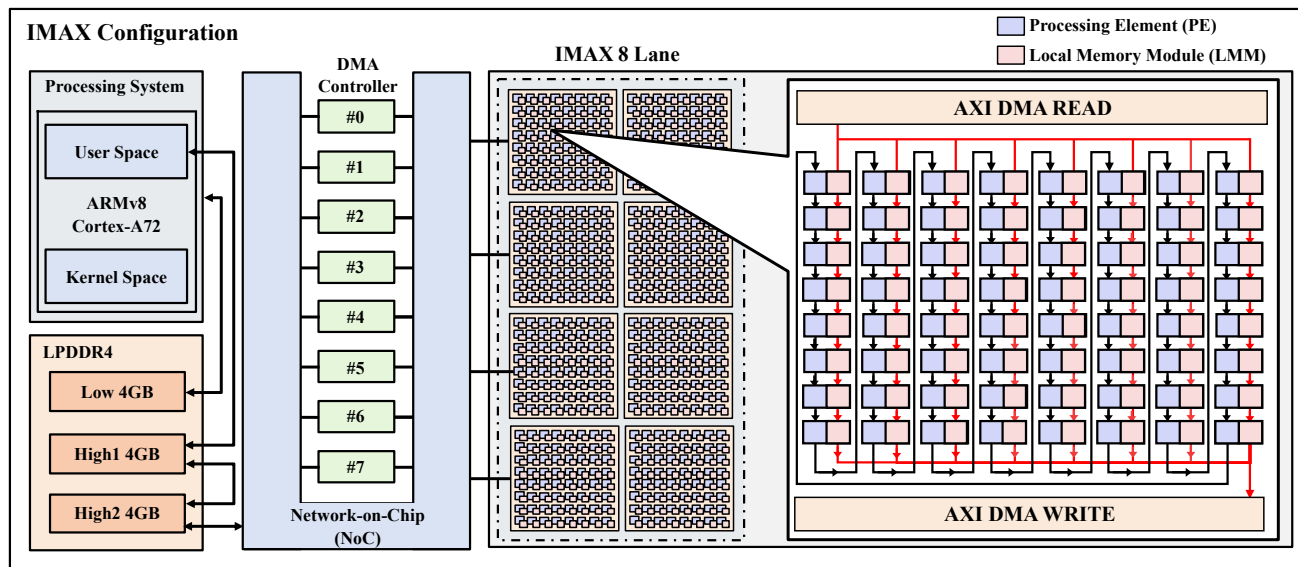


図2 IMAX3 プロトタイプの構成図. PS 上のホスト CPU と PL 上の CGLA が協調して動作する. 各 IMAX レーンは DMA コントローラを介して接続され、レーンの追加により構成を拡張できる. 各レーン内では PE 間でデータを直接受け渡ししながらパイプライン的に動作し、演算を進める.

表1 共通する実装と初期評価の整理.

対象アプリ	共通化できる部分	初期評価で得られた結果	残る課題
LLM (llama.cpp) [1],[13],[14]	CPU 側の行列積・量子化 dot 積を IMAX にオフロードし、ホスト側が制御フローと KV キャッシュ管理を担当する.	Q6_K 実装でエンドツーエンド実行時間が 34.2s から 30.4s に短縮した. 動的チャンク分割によりトークン解析フェーズは 1.62× 高速化した. Qwen3 0.6B Q8 では DRAIN 帯域が 0.264 Gbps から 1.077 Gbps に改善し、エンドツーエンド実行時間は 97.2s から 87.2s に短縮した.	トークン生成フェーズでは LOAD とホスト側処理が支配的であり、チャンク方策, LOAD/EXEC/DRAIN の各段階を分けた計測が重要である.
音声認識 (whisper.cpp) [12]	FP16 dot 積を IMAX にオフロードし、残差処理と前後処理はホスト側に残す混合実行を採る.	Whisper-tiny.en で Jetson AGX Orin 比 1.90×, RTX 4090 比 9.83× の電力効率を達成した. EXEC 比率は 60~75% であった.	FP16/FP32 混合実行を前提にしたバックエンド設計と、ローカルメモリモジュール (local memory module: LMM) サイズ探索, LOAD/EXEC/DRAIN の各段階の計測が必要である.
画像生成 (stable-diffusion.cpp) [11]	LLM 向けに整備した量子化 dot 積カーネルを再利用し、U-Net の一部計算を IMAX にオフロードする.	dot 積実行時間の内訳は FP32 30.7%, FP16 59.0%, Q3_K 10.3% であり、非量子化演算の比率が大きかった.	データ型ごとにバックエンドを切り替える必要があり、非量子化カーネルの拡充が今後の課題である.

列実行する. これに対し CGLA では、各 PE の近傍に LMM が配置されるため、演算器とメモリの距離が短く、中間データを隣接 PE/LMM 間で直接受け渡ししやすい [7], [8]. この近接メモリ配置と直接受け渡しにより、IMAX では演算段階を PE 列に沿って並べたパイプラインとして段階的カーネルを実装する.

図2にIMAX3実機構成を示す. IMAX3実機は、Arm Cortex-A72を含むProcessing System (PS)と、8 laneのCGLA本体を実装するProgrammable Logic (PL)からなるSoCである. 評価ではVPK180を4枚用い、そのうち1枚をホスト、残る3枚をスレーブとして用いる. アプリケーションはホスト側で

実行を制御し、DMAを通じてデータをLMMへ搬送した後、CGLA上でカーネルを実行する. 各 lane は 64 個の PE と対応する LMM から構成され、PE と LMM は一次元に交互配置される [8]. lane 内には、PE から次段 PE へ部分和を渡す実行データパスと、各 PE が隣接 LMM を参照するメモリデータパスが並行して存在する. 各 PE は演算器群とアドレス生成器を持ち、LMM は DMA と組み合わせたダブルバッファを前提とするため、dot 積やバタフライ演算のような段階的カーネルを PE 列に沿ったフィードフォワード型パイプラインとして実装しやすい [8]. 一方でエンドツーエンド実行では、ホスト側

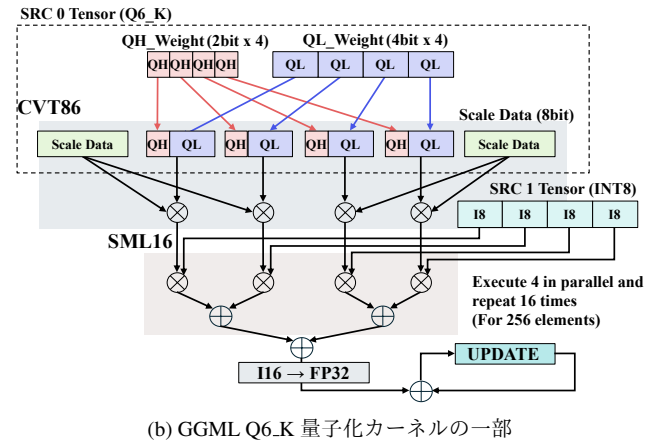
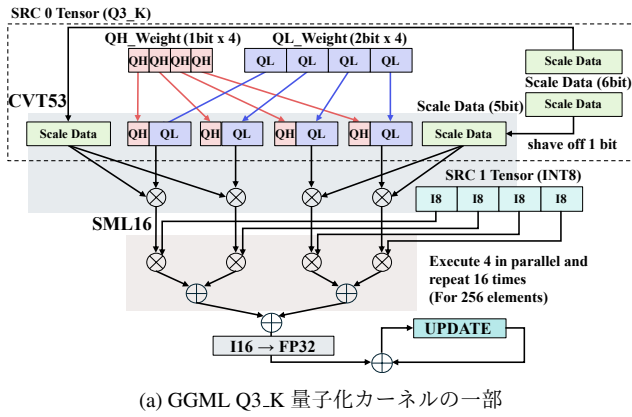


図3 LLM実装で用いられた量子化 dot 積カーネルのデータフロー例。実際には PE 内で論理 4 列のマルチスレッディングで実行される。いずれも量子化値とスケールを読み出しながら dot 積の直前で復号し、最終的に FP32 の部分和へ集約するが、重みブロックの構成とスケール幅が異なる。

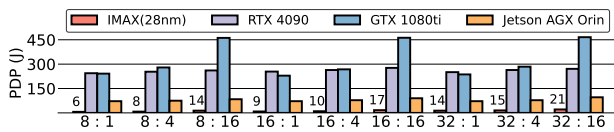


図4 Qwen3 0.6B Q3_K.S における各計算基盤での PDP 比較。IMAX (28 nm 推定), RTX 4090, GTX 1080 Ti, Jetson AGX Orin を比較する。

がチャンク分割, DMA 起動, 結果回収, 次のカーネル呼び出しを明示的に制御する必要がある。特に, open, LMM 確保, mmap, ioctl による起動・完了待ちの手順は三つのアプリで共通化しやすく, オフロードする演算境界とデータ型変換はアプリ依存の部分として残る。

既報では, IMAX 上で SpGEMM と FFT が実装され, CGLA の基本データフローが示されている [8]。その後, LLM, 音声認識, 画像生成へと適用範囲が広がり [1], [11], [12], 一次元 CGLA を前提としたカーネル設計とホスト側制御がアプリごとに蓄積されてきた。

表 1 に, 共通する実装と初期評価の要点をまとめる。表ではアプリごとに共通化できた実装, 得られた結果, 残る課題を並べ, 本文ではそれぞれの支配演算とデータ型の違いを述べる。

3.2 LLM 実装と評価

LLM 実装では llama.cpp を基盤として, Q8.0, Q3_K, Q6_K などの量子化 dot 積をオフロードしてきた。三つのアプリに共通するカーネル実行の枠組みは次のとおりである。入力ブロックと重みブロックを LMM に読み込み, 必要な復号やデータ型変換を積和の直前で行う。その後, PE 列で SIMD 的に積和し, 列方向のマルチスレッド化で複数の出力列を並列に処理し, 最終的に FP32 の部分和または出力として返す。LLM では, この枠組みの中で量子化ブロックの復号と dot 積を組み合わせる。図 3 に, LLM 実装で用いた Q3_K と Q6_K の量子化 dot 積カーネルを示す。ここでは量子化値とスケールを読み出しながら dot 積の直前で復号し, 対応するベクトルとの積

和を進め, 最終的に FP32 の部分和として出力する。したがって, 量子化カーネルの設計点は, どの単位で量子化ブロックを LMM に置か, どの段で復号するか, どこまでを PE 列上で連続実行するかにある。Q8.0, Q3_K, Q6_K では, 量子化ブロックの構成, スケールの配置, 復号手順がそれぞれ異なるため, 同じ dot 積でも量子化形式ごとに別のカーネル実装が必要になる。

図 4 に, Qwen3 0.6B Q3_K.S における入力/出力 token 数の組ごとの PDP (Power-Delay Product) を示す。ここでは IMAX (28 nm 推定), RTX 4090, GTX 1080 Ti, Jetson AGX Orin の 4 基盤を同一の量子化形式で比較している。PDP はレイテンシと消費電力を掛け合わせた指標であり, 図 4 は入出力長の組に応じて, カーネル本体だけでなくホスト側処理と転送を含む全体の効率がどう変わるかを示している。したがって, 図 4 の比較は, 単一カーネルの実行速度だけでなく, 量子化 dot 積を既存 OSS の制御フローへ組み込んだときのエンドツーエンド実行効率を見るための比較として位置付けられる。図に含まれる Qwen3 0.6B Q3_K.S [32:16] workload では, 総レイテンシ 16.3 s のうち IMAX カーネル実行は 4.47 s (27.4%) であり, ホスト CPU 処理 5.43 s (33.3%) と DMA による入力転送 5.31 s (32.6%) が大きな割合を占める [1]。また, トークン解析フェーズでは複数トークン分の計算をまとめて流しやすいのに対し, トークン生成フェーズでは 1 トークンごとの呼び出しと転送の固定費が前面に出るため, カーネル本体だけでなくチャンク分割と出力転送のまとめ方が全体性能に直結する [1], [14]。ここでの方針は, 既存 OSS 全体を作り直すのではなく, 制御フローや KV キャッシュ管理はホスト側に残し, 計算量の大きいカーネルのみを IMAX にオフロードすることである。以降の音声認識と画像生成では, この枠組みに対して支配的なデータ型と後段処理の違いを見る。

ここでいう段階別計測とは, LOAD, EXEC, DRAIN など各段階の時間や転送量を分けて記録する計測である。トークン解析フェーズとトークン生成フェーズを分け, さらに各段階を分解して観測することで, どこが律速になっているかを追

える。また、量子化形式と LMM 容量に応じてチャンクサイズを切り替えるランタイム方策が必要である。動的チャンク分割による 1.62 倍のトークン解析フェーズ高速化[13]は、ランタイムの改善が全体性能に大きく影響することを示している。この傾向は、LLM 推論でメモリ管理と実行方策が性能を左右することを示した PagedAttention/vLLM とも整合する [9]。

3.3 音声認識と画像生成への展開

Whisper 音声認識実装では、量子化復号の代わりに FP16-to-FP32 変換を積和の直前に行い、PE 列での SIMD 実行と列方向のマルチスレッド化を組み合わせる [12]。LLM との差は、重み量子化 dot 積が中心になるのではなく、FP16 の入力を用いた積和と、その後段の FP32 処理が中心になる点にある。Whisper 系ワークロードでは、LLM よりも前後処理や制御処理がホスト側に残りやすく、完全オフロードではなく混合実行が自然な構成になる。このため、量子化カーネルだけでは不十分であり、FP16/FP32 のデータ型を横断する API、残差処理のホスト実行、およびバースト設定と LMM サイズ探索を支えるプロファイリングが必要である。Whisper-tiny.en での Jetson AGX Orin 比 1.90 倍、RTX 4090 比 9.83 倍の省エネ結果 [12] より、エッジ側 CGLA では低レイテンシなランタイムと混合バックエンドを前提に設計する必要がある。

Stable Diffusion 実装では、Q3_K などの量子化 dot 積を再利用できる部分がある一方、U-Net のその他の部分では FP16 や FP32 の演算が支配的である [11]。実行時間の内訳は FP32 30.7%、FP16 59.0%、Q3_K 10.3% であり、量子化カーネルだけを拡充しても全体最適にはならない [11]。したがって、画像生成まで含めるには、量子化 dot 積の再利用に加えて、FP16/FP32 演算をデータ型ごとにバックエンドへ振り分ける必要がある。すなわち、U-Net 内でも量子化 dot 積として切り出せる部分には LLM 向けカーネルを流用できるが、それ以外の FP16/FP32 主体の部分はホスト側処理として残る。ここで問題になるのは、同じ CGLA ハードウェアを使っているにもかかわらず、アプリごとに支配的なデータ型とホスト側に残る処理が異なることである。そのためソフトウェアスタックの役割は、単に既存カーネルを再利用することではなく、テンソル境界、転送単位、プロファイリング点を共通化しつつ、どのデータ型をどのバックエンドへ送るかを制御する振り分け層を持つことにある。Whisper と Stable Diffusion の実装差分は、バックエンド本体を完全共通化するよりも、混合実行とデータ型混在を前提にしたフロントエンドとディスパッチ層を整える必要があることを示している。

3.4 共通課題

表 1 で整理した三つのアプリを比較すると、共通課題は、転送とホスト協調、データ型混在、Python OSS と C/C++ 実装の橋渡しに集約される。このうち、アプリ横断で再利用しにくさが最も表れやすいのは、第 3 の橋渡し部分である。

現行の CGLA 実装は C/C++ に埋め込む形をとる一方、多くの AI OSS は Python を入口としており、モデル記述、前後処理、推論制御は Python 側に残る。そのため、新規アプリごとに、演算の切り出し、LMM 容量を意識した分割、段階別計測の取得、CPU 実行と CGLA 実行の切替を個別に記述する必要

表 2 Python OSS と C/C++ 実装の間で個別対応になりやすい点。

観点	Python OSS 側で扱われる内容	現行 CGLA 実装で個別対応している内容
演算呼び出し	演算グラフ、モジュール呼び出し、推論ループ	dot 積や行列演算への手作業マッピング
メモリ管理	Tensor object と自動確保/解放	LMM 容量を意識した分割と DMA 制御
計測取得	プロファイラ用フック、ログコールバック	段階別時間計測、出力転送回数、チャンク記録
実行切替	Python 側でのバックエンド指定	ホスト CPU 実行と CGLA 実行の分岐実装

が生じる。Whisper で観測されたように、dot 積以外の処理がホスト側に残る場合、問題は単なるカーネル呼び出し API ではなく、どの演算をオフロードし、どの計測点を共通化するかへ移る。表 2 は、この分断がどこで生じるかを、Python OSS 側で自然に扱われる内容と、現行 CGLA 実装で個別対応している内容の対応として整理したものである。したがって、共通化すべき対象は、テンソル境界、転送、計測形式であり、バックエンド本体はデータ型とアプリに応じて分かれる。

4. 考察と今後の展望

4.1 既存実装に共通する基盤と今後の改善

LLM では、llama.cpp 系実装の改善が中心課題になる。優先度が高いのは、量子化形式ごとのカーネルを増やすことと、量子化形式、テンソル形状、トークン解析フェーズ/トークン生成フェーズ、LMM 容量に応じて経路を選ぶディスパッチを整備することである。同じオフロード構成でも、どの量子化形式をどの条件で IMAX に送るかによって、エンドツーエンド性能とデータ移動コストは変化する。量子化形式の選択が性能とメモリ使用量を左右する点は、一般の LLM 量子化研究とも共通する [10]。したがって、既存実装の改善は、新演算命令の追加だけでなく、カーネル選択条件とデータ移動制御を一体で見直す課題になる。

4.2 ログとランタイム方策設計

IMAX の計測基盤では、DRAIN、CONF、REGV、LOAD、EXEC の実行時間を分離表示できる。このため、どのカーネルを選ぶか、チャンクをどの粒度で切るか、出力転送や DRAIN をどこでまとめるかを、段階別ログに基づいて判断できる。継続的に収集すべきなのは、DMA 読み込み時間、出力転送時間、チャンクごとの待ち時間、トークン解析フェーズ/トークン生成フェーズの分離ログである。

エッジ構成では、小さな LMM 容量に合わせたチャンク設定、ホストとアクセラレータの混合実行、低レイテンシな DRAIN 制御が優先される。一方、サーバ構成では、高帯域メモリを導入しても、テンソル配置、チャンク粒度、要求並列度を同時に設計しなければ固定ボトルネックが残る [9], [14]~[16]。したがって、将来の Xeon+PCIe 系や 3D 積層 DRAM 系でも、必要なのは単純な帯域増加ではなく、メモリ階層を前提にしたラ

ンタイム設計である。

4.3 今後の研究課題

今後の研究課題は三点である。第1に、量子化形式ごとのカーネル拡充とディスパッチ条件の整理である。第2に、適応的なチャンク分割と帯域考慮スケジューリングを固定方策と比較し、LLM、音声認識、画像生成で方策の一般性を評価することである。その際、token/s や実時間比だけでなく、チャンク回数、出力転送平均サイズ、LOAD/EXEC 比率といったスタック内部指標も同時に報告する必要がある。第3に、ディスパッチ、転送、プリフェッチの判断を後から検証できるように、段階別時間、転送量、待ち時間、バッファ使用量を継続的に記録する設計基盤の整備である。将来の3D 積層 DRAM 対応では、重み配置、KV キャッシュ配置、チャンク単位、転送単位を含めてメモリ階層を意識した基盤設計が必要になる。

5. おわりに

本稿は CGLA 上の LLM、音声認識、画像生成を扱うソフトウェアスタック研究の位置づけを整理した。表1と第3章の比較から、ホスト側制御、DMA 転送、段階別計測といった骨格はアプリ間で共通化できる一方、量子化形式、混合精度、前後処理の残り方に応じてアプリ依存部分が残ることを整理した。特に LLM では量子化 dot 積とチャンク方策、音声認識では FP16/FP32 混合実行、画像生成では非量子化演算の比率が主要因となり、単一の共通バックエンドだけでは吸収できないことを確認した。今後の課題は、llama.cpp 系での量子化カーネル拡充とディスパッチ条件の整理、ランタイム方策の定量評価、3D 積層 DRAM 世代への構成検討である。

謝 辞

本研究の一部は JST 戦略的創造研究推進事業先端的カーボンニュートラル技術開発 (ALCANext) JPMJAN23F4 の支援を受けたものである。また、本研究は東京大学 VDEC 活動を通して、日本シノプシス合同会社の協力で行われたものである。

文 献

- [1] T. Ando, Y. Eto, A. Takeuchi, and Y. Nakashima, "Efficient Kernel Mapping and Comprehensive System Evaluation of LLM Acceleration on a CGLA," *IEEE Access*, vol. 13, pp. 199631–199646, 2025.
- [2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, and R. Boyle et al., "In-datacenter performance analysis of a tensor processing unit," *Proc. ISCA*, pp. 1–12, 2017.
- [3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
- [4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [5] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," *Proc. ASPLOS*, pp. 461–475, 2018.
- [6] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A Reconfigurable Architecture for Parallel Patterns," *Proc. ISCA*, pp. 389–402, 2017.
- [7] A. Podobas, K. Sano, and S. Matsuoka, "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective," *IEEE Access*, vol. 8, pp. 146719–146743, 2020.
- [8] T. Akabe, V. T. D. Le, and Y. Nakashima, "IMAX: A Power-Efficient Multilevel Pipelined CGLA and Applications," *IEEE Access*, vol. 13, pp. 31899–31911, 2025.
- [9] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," *Proc. SOSP*, pp. 611–626, 2023.
- [10] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," *Proc. ICML*, pp. 38087–38099, 2023.
- [11] T. Ando, Y. Eto, and Y. Nakashima, "Implementation and Evaluation of Stable Diffusion on a General-Purpose CGLA Accelerator," *Proc. MCSoc*, pp. 752–759, 2025.
- [12] T. Ando, Y. Eto, A. Takeuchi, and Y. Nakashima, "Energy-Efficient Hardware Acceleration of Whisper ASR on a CGLA," *Proc. CANDAR*, pp. 85–91, 2025.
- [13] T. Ando, A. Takeuchi, Y. Eto, Y. Munakata, and Y. Nakashima, "Q-Snap: Quantization-Aware Dynamic Chunking for LLM Execution on a CGLA," *Proc. ICISN*, 2026.
- [14] T. Ando, Y. Eto, and Y. Nakashima, "A Detailed Analysis of LLM Execution on IMAX3 and Initial Evaluation of IMAX4 Prototype for Server Environment," *Proc. SASIMI*, pp. 8–13, 2025.
- [15] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM Technology and Architecture," *Proc. IEEE International Memory Workshop*, pp. 1–4, 2017.
- [16] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," *Proc. ASPLOS*, pp. 751–764, 2017.