修士論文

CGRA 向け疎行列積の高速化手法と評価

菅原　琢哉

2022 年 3 月 18 日

奈良先端科学技術大学院大学
情報科学研究科 情報科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士（工学）授与の要件として提出した修士論文である．

菅原　琢哉

審査委員：

中島　康彦 教授 （主指導教員，情報科学領域）
林　優一 教授 （副指導教員，情報科学領域）
張　任遠 准教授 （副指導教員，情報科学領域）

CGRA 向け疎行列積の高速化手法と評価*

菅原　琢哉

内容梗概

近年, 機械学習や SLAM などの言葉が注目を集めている. それらのアプリケーションでは, 疎行列積がボトルネックとなっており, 複雑な疎行列積計算を高速に処理するハードウェアが必要とされている．従来, 疎行列演算を処理する場合は特定の format に行列を格納し, ソフトウェア最適化を行うことで汎用計算機を高速化をするのが定石であった．ハードウェアに疎行列計算機構を入れて汎用性を落として性能をあげるよりも, 半導体微細化や動作周波数向上によってハードウェア性能向上を期待する方が合理的であったからである．しかし, 半導体微細化や動作周波数の向上をこれ以上望むことができない今, 汎用計算機にこれ以上の性能向上を期待するのは難しい．そこで, 疎行列計算機構をハードウェアに取り入れる専用ハードウェア (Domain Specific Architecture: DSA) や Field Programmable Gate Array (FPGA) による実装方法が数多く報告されているものの, 前者は汎用性に欠け, 後者は動作周波数が一桁低いという構造的問題を抱えている．上述の問題を解決できるアーキテクチャとして,Coarse Grain Reconfigurable Array (CGRA) 注目されている． ただし, 従来型 CGRA では疎行列計算に必須であるアドレス計算機構などを持ち合わせておらず, 密行列のみにしか適用できない問題があった. そこで, アドレス計算機構を持つ In-Memory Accelerator eXtension (IMAX) を疎行列計算に利用することが期待される．本研究では IMAX に適した疎行列積の実装方法を提案した．IMAX 専用の疎行列 format を Index-Value-Set(IVS) 法と Index-Value-Joint(IVJ) 法の二通り提案し,IVJ 法の優位性を確認した後に IVJ 法に適した疎行列積の実装を行った．密行列積 1024×1024 を従来手法で行った実行時間と比較して,Sparsity が 95% の際に計算時間を 92.9% 削減することを確認した．また,10 個の実データセットを用いて疎行列ベクトル積をベンチマークを行い,

*奈良先端科学技術大学院大学 情報科学研究科 情報科学専攻 修士論文, NAIST-IS-MT2011145, 2022 年 3 月 18 日.

平均で計算時間を 94.3% 削減することに成功した.

キーワード

CGRA, アクセラレータ, エッジコンピューティング, CSR, 疎行列積計算

Acceleration Method and Performance Evaluation of Sparse Matrix Products on CGRA*

Sugahara Takuya

Abstract

In recent years, terms such as machine learning and SLAM have been gaining attention. In these applications, sparse matrix products have become the bottleneck, and there is a need for hardware that can process complex sparse matrix product calculations at high speed. Conventionally, when processing sparse matrix operations, it is standard practice to store the matrices in a specific format and perform software optimization to speed up general-purpose computers. This is because it was more reasonable to expect hardware performance improvement through semiconductor miniaturization and higher operating frequency, rather than to reduce generality and improve performance by adding a sparse matrix calculation mechanism to hardware. However, now that we cannot hope for further miniaturization of semiconductors and improvement in operating frequency, it is difficult to expect further performance improvement in general-purpose computers. Therefore, a number of implementation methods using dedicated hardware (Domain Specific Accelerator (DSA)) or Field Programmable Gate Array (FPGA) have been reported. However, the former lacks versatility, and the latter has a structural problem in that its operating frequency is lower by an order of magnitude. The Coarse Grain Reconfigurable Architecture (CGRA) has been attracting attention as an architecture that can solve the above-mentioned problems. However, conventional CGRAs do not

*Master's Thesis, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT, March 18, 2022.

have an address calculation mechanism, which is essential for sparse matrix calculation, and can only be applied to dense matrices. Therefore, we expect that In-Memory Accelerator eXtension (IMAX), which has an address calculation mechanism, can be used for sparse matrix calculation. In this study, we propose an implementation method of sparse matrix product suitable for IMAX. We also proposed two sparse matrix formats specifically for IMAX and implemented a sparse matrix product suitable for the proposed format. Compared with the running time of the dense matrix product 1024×1024 with the conventional method, we confirmed that the proposed method reduces the computation time by 92% when the sparse matrix fraction is 95%. We also benchmarked the sparse matrix-vector product on 10 real data sets and found that the computation time was reduced by 99.5% on average.

Keywords:

CGRA, accelerator, edge computing, CSR, sparse matrix product computing

目次

図目次

第 1 章　はじめに

1.1　背景

近年, 疎行列計算が多くのアプリケーションで必要とされている．DNN (Deep Neural Network) は多くの産業に応用されているが, 計算量がボトルネックとなっており, 組み込み環境などの制約下でリアルタイム処理ができない問題が指摘されている．そこで,DNN の重みが疎行列であることに着目し, 計算量を減らす取り組みが報告されている.[6] また,DNN の不要な重みを除く実験を行う宝くじ仮説は,DNN の本質に迫る仮説として注目されており, 疎行列の重要性は増している.[7] DNN の中でも Attention は特に応用範囲が広く, 自然言語処理や画像処理などに応用されているが, ボトルネック部分の計算量が $O\left(N^3\right)$ であり, 疎行列を工夫して計算する必要がある．また,Attention の重み部分が疎になるように学習する手法も報告されている.[8]
DNN の他に疎行列計算が必要な例として,SLAM (Simultaneous Localization and Mapping) が挙げられる.[9] SLAM の応用先として自動運転や自立移動型ロボットなどが挙げられるが, それらのアプリケーションは組み込み環境下でリアルタイム性が求められており, 疎行列を工夫して計算する必要がある．一例として, 疎行列の規則性に着目して, 行列分解後も疎行列を維持する例などがある.[10]
これまで述べた計算量削減だけでは, リアルタイム性を達成できないことが多く, 疎行列計算のハードウェア的高速化も必要とされている．しかし, 汎用計算機だけで DNN や SLAM の疎行列計算を処理するのは限界があり, 専用ハードウェアが必要とされている．DNN や SLAM の疎行列部分を高速化するハードウェアとして Graphics Processing Unit (GPU) が多く使われているが, 面積効率や消費電力の観点で組み込み環境には向かない.[11][9] DSA を用いて疎行列計算を高速化する例もあり [5], デザイン次第で低消費電力が見込めるが, 汎用性が低く, 理論の進化速度が速い DNN や SLAM に適応するのには向いていない．FPGA は GPU と比較して消費電力が低く,DSA よりも汎用性があるが, 動作周波数の低さやコンパイル時間が問題となっている．従来型の CGRA では, 疎行列計算の際に必要なアドレス計算機構を持ち合わせておらず,CGRA の利点を生かした計算が困難である．消費電

力が低く, 汎用性があり, アドレス計算機構を持ち合わせたハードウェアが必要とされている.

1.2 目的

高い電力あたり性能を持つアクセラレータとして CGRA が研究されている. CGRA の一種であるシストリックアレイは DCNNs においても高い電力あたり性能を持つことが示されている.[12] DCNNs だけでなく, ステレオ画像生成や距離の取得にも応用可能な Light-field 画像処理の高速化を行うことができるアクセラレータとして, 我々は CGRA アクセラレータ EMAXV を提案してきた. そして,EMAXV の演算ユニットを時分割 4 重実行と浮動小数点演算パイプラインの導入によって, 性能低下を抑えつつ 1 次元構造化 (演算器数と局所記憶数は 1/4) を可能にしたリニアアレイアクセラレータである IMAX を提案してきた [13]. しかし, 疎行列計算による実装および評価は行われておらず,DNN などのアプリケーションで本来の性能を発揮でいなかった. また行列積を計算する際に Local Memory(LMM) を十分に利用できておらず, さらに毎回人手で転送範囲をチューニングする必要があった. 本研究では IMAX による疎行列計算の実装を行う. また, 当初の疎行列計算を実装する計画に加え, 多くのテストパターンを試すために LMM のスケジューリングを自動化することも目的とする

1.3 構成

本章では, 本研究の背景と目的を述べた. 2 章に関連研究を示し, 3 章にて従来型の CGRA の課題と研究対象とする CGRA を説明する. 4 章に 疎行列積スケジューリングおよび LMM の利用率を改善する提案手法を示す. その後,5 章に FPGA によるプロトタイプシステムと疎行列積の評価を行い, 最後に,6 章を本論文のまとめとする.

第 2 章　関連研究

エッジデバイス向けの低コストかつ省電力な疎行列アクセラレータが求められているが, 従来型のノイマン型計算基盤だけでは対応が困難であることを述べた．そこで, 本章ではノイマン型計算基盤以外に現在幅広く用いられている GPU と DSA を説明したのち, 現行システムにおける DSA の問題点について説明する．そして, 我々が着目しているアーキテクチャである CGRA ついて説明する．

2.1　GPU

汎用プロセッサはスーパースカラによって拡張された複数演算の同時実行機構の演算器使用率をさらに高めるため Out-of-Order(OoO) 実行機構と, 複雑な分岐を持つコンテキストを高速に実行するため投機実行機構と分岐予測器を備えている．これらの機構は Re-order buffer(ROB) や分岐ヒストリテーブルといった多くのレジスタを必要とする回路が必要なため, 面積的にも消費電力的にもオーバヘッドが大きい．しかし, 背景で述へたような DCNNs や規模の大きい画像処理などは, ほとんどがループのアンローリングが可能な複雑な分岐を持たないプログラムであり,OoO 実行機構や分岐予測器の恩恵をほとんど受けない．ここでいう OoO 実行機構の恩恵を受けないの意味は, DCNNs のようなアプリケーションでは並列に実行可能な命令は静的に解決されるため, ROB が必要ないことを言っている．それぞれの演算ユニットは Single Instruction Multiple Data(SIMD) 演算を行う．複数の演算ユニットは 32 コアで一つのグループとして扱われ, 共通の命令が発行される．GPU メーカーの最大手である NVIDIA はこのグループを Streaming Multiprocessor(SM) と呼び, 複数演算ユニット に同一の命令を発行する手法を Single Instruction Multiple(SIMT) 方式と呼んでいる．簡単な分岐は条件付き実行命令でサポートしている．それぞれのコアで走るコンテキストはスレッドとして表現され, SM に発行されるスレッドの束は Warp や Wavefront と呼はれる．GPU は ROB を持たないが Load 命令などで Warp がストールすると演算器稼働率が低下するためディスパッチャが異なる Warp を割り当て演算器を稼働させる．GPU は SIMT 方式とディスバッチャによって膨大な数の演算器を高い稼働率で動作さ

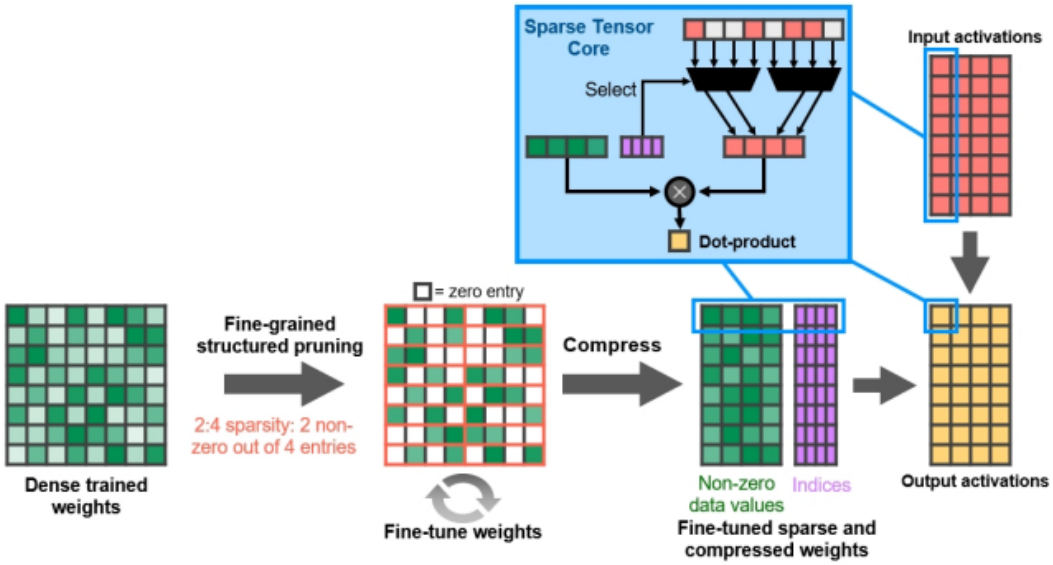

図 2.1　Sparse Tensor Core[1]

せることが可能である．しかし, Warp のストールによるペナルティを隠蔽するためにコンテキストスイッチが必要であるため, 深いパイプラインを持った演算器を持つことが不利になる．そこで, 膨大な数の演算器のデータ供給は莫大な広さを持つ主記憶帯域とデータのバースト転送効率を上げるためのコアレッシング機苒, および大きなレジスタファイルとキャッシュによりカバーする．これらの機構のデータフローをハードウェアが制御する．さらに Warp はそれぞれプログラムカウンタを持つため, プログラミングが比較的容易である．しかしながら, 組み込み機器の実装は面積, 電力面でオーバヘッド となるためトレードオフの関係である．そこで近年は DCNNS の学習を GPU で行い, 学習結果の重みパラメータを用いて推論のみを低消費電力で高速に行う DSA か数多く提案されている．また近年, 疎行列演算を高速化するために Sparse Tensor Core というものを搭載しており, value から Non Zero の場所だけを index で選び取り効率的に計算している.[1] 図 2.1 の例では 8 要素から 4 要素だけをハードウェア内で選び取り理論上二倍のスループットを実現している．Sparse Tensor Core は従来の Float 32bit のみならず独自規格の TF32 や INT8 を搭載しており, DNN の推論高速化を実現している．

2.2 FPGA

FPGA は回路を何度でも変更可能な再構成可能デハイスである．SRAM によって構成され, 論理ゲートとして振る舞う Logic Block, 外部との I/O を行う I/OBlock, それらの接続を行う Connection Block が FPGA の基本的な構成要素である．かつてはその動作周波数の低さから, 専ら ASIC のプロトタイプ開発や, 性能を要求されないグルーロジックや万能 IC として用いられていたが, 半導体プロセス技術の進歩に伴い，集積度と動作周波数が向上し，DSP が搭載されているものも多く, 近年では計算用デハイスとしても活用されている. 主記憶アクセスを最小限に抑える専用回路を構成できることから, CPU や GPU に比べて電力効率は大幅に優れているが, ASIC として製造された DSA には及ばない．しかしながら, その汎用性から大量生産が見込めるため, LSI 製造の初期費用の回収が比較的容易である．[2] では疎行列に特化した回路を図 2.2 のように FPGA 上で実現するしており,GPU に比べて低い消費電力と低いメモリバンド幅で Sparse matrix-vector multiplication(SpMV) の高速化に成功している．しかし, 最新の FPGA では既に先端プロセスを使っており, 微細化による性能向上はこれ以上期待できないため, Fpga ベンダーの Xilinx は DSA と FPGA の欠点を補い合うようなアーキテクチャを提案しており, DNN 向けの DSA が DNN のメイン計算を行い, 周りの FPGA が IO を担うことで, 高速かつ低遅延の演算を実現している.[14]

2.3 DSA

DNN の推論処理やデータベース処理など特定利用のための専用ハードウェア (DSA) を開発し, アプリケーションを高速化する取り組みが流行している．DSA の例として, Eyeriss v2[15],FlexFlow[16],Graphcore[17],OuterSPACE[3] などがあげられる．Eyeriss v2 や FlexFlow などの推論処理用 DSA の特徴として, DNN の推論は演算の精度を落としても, 認識の正誤率はそれほど下がらないことが示されているため, 重みの情報量を削減し 16bit − 8bit の固定小数積和演算を行うことで演算性能, 面積効率および電力効率を向上している．Graphcore のように DNN の学習時間を減らす DSA も存在し,GPU に比べて対費用効果で上回っている．ここで,OuterSPACE の外積版行列積の概略を図 2.3 に示し,OuterSPACE の概要図を

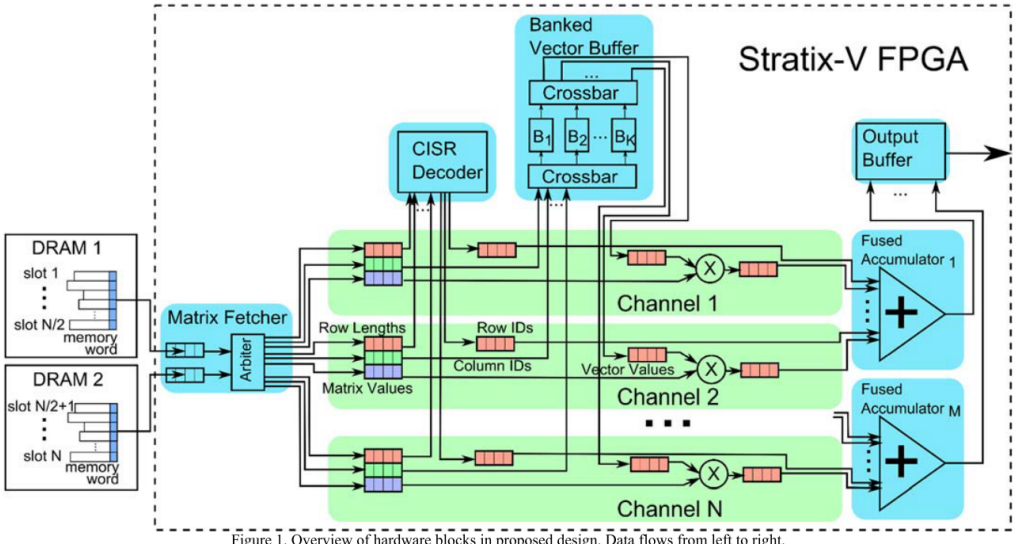

Figure 1. Overview of hardware blocks in proposed design. Data flows from left to right.

図 2.2　FPGA-based SpMV[2]

図 2.4 に示した．OuterSPACE のように疎行列積を外積で計算し疎行列計算でボトルネックとなる index 計算時間増加, 不規則な Read Write, メモリ再利用率の低下などを防いでいる．しかし,OuterSPACE も他の DSA と同様に汎用性の低下を避けることができない．トレンドになっているようなアルゴリズムは進歩が速く,ASIC が完成したころには既に使えなくなっている恐れがある．他の OuterSPACE の欠点として, 外積計算したことによる局所性低下である．Multiply のあとに Add が必要で Output を Link List で管理する必要があるので, ボトルネックとなっている．

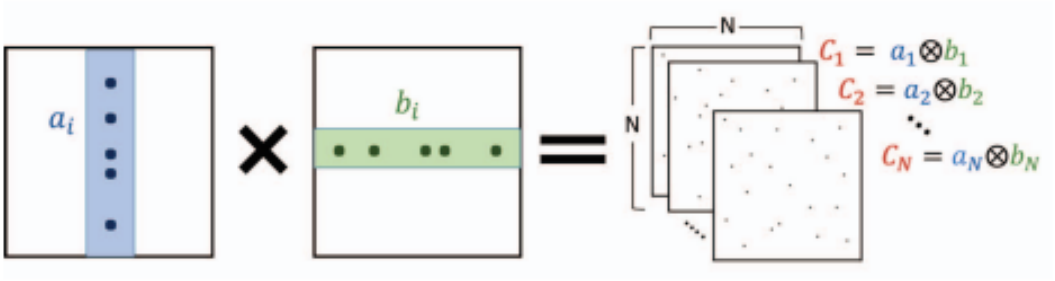

図 2.3　outer product[3]

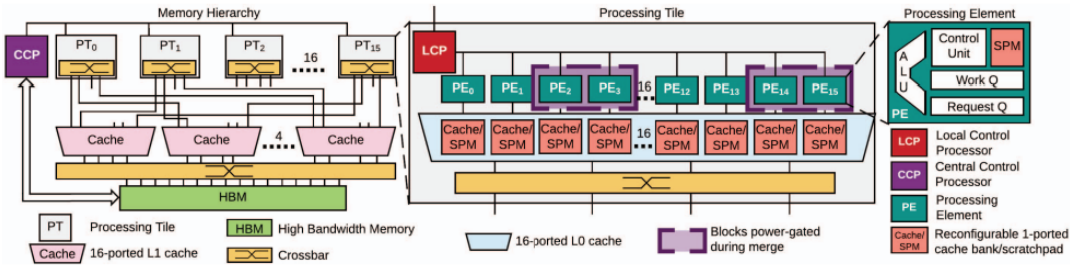

図 2.4　OuterSPACE[3]

2.4 CGRA

CGRA はシストリックアレイ [4] を起源とするアーキテクチャで, FPGA と同様に再構成可能なアーキテクチャである．GPU より高い電力当たり性能を達成する計算デハイスとして CGRA は注目されている．図 2.5 に一般的な CGRA の構造を示す．CGRA は演算器およびレジスタ等からなる基本ユニットとメモリから構成されることが多い．再構成の粒度が FPGA ではゲートレベルであるのに対し, CGRA では演算器レベルであり, 同等プロセスルールで同等機能を実現する場合, FPGA に比べて動作周波数や回路面積で優位である．さらに, FPGA を用いた開発では,Register Transfer Level(RTL) 記述からゲートレベルに合成したネットリストを実際の FPGA の構造に落とし込む配置配線のプロセスがあり, 大規模な最適化問題を解く必要がある．回路の規模と制約によっては合成と合わせて数十分から数十時間を要する．一方で, CGRA では上述のように再構成の粒度が粗く, 探索範囲が大幅に削減できるため, コンパイル時間は数秒から数十秒程度である．そのため, CGRA は FPGA に比べて開発における turnaround time の面でも優位である．また, CGRA は GPU のような SIMD 型水平方向並列処理ではなくパイプライン型垂直方向並列処理であり, 演算器群に毎サイクル供給すべきデータを抑えることができる．このため, 大規模なメモリパスやコアレシング機構は不要であり, メモリバス幅を確保しづらいエッジにおいて有望なアーキテクチャである．に CGRA アクセラレータの例として ADRES[18] のインスタンス例を示す．ADRES は VLIW view と CGA view から成る密結合アーキテクチャとなっている．VLIW contorl unit (CU) が CGA のループ制御を VLIW の関数コールとして行う．CCGA 実行時には, データの通信は VLIW Section の Functional Unit (FU) を通じて行う． ADRES の他に CGRA アクセラレータとして, PipeRench[19], CMA[20], LAPP[21] 等が提案されている．また, シストリックアレイをコアとなる行列積エンジンの アーキテクチャに採用した Google の TPU[12] は, 高い面積あたり性能および電力効率を達成し, 話題となったことは記憶に新しい．複数の CGRA を用いた関連研究としては, 2017 年に開催された Hot Chips29 において Wave Comuting 社が発表した Dataflow Processing Unit (DPU)[22] があげられる．DPU はクラスタ化を前提とした高性能計算基盤向け CGRA アクセラレータであり, 64-DPU のクラスタで AlexNet の学習を 40 分で完了できると報告されている．本研究にお

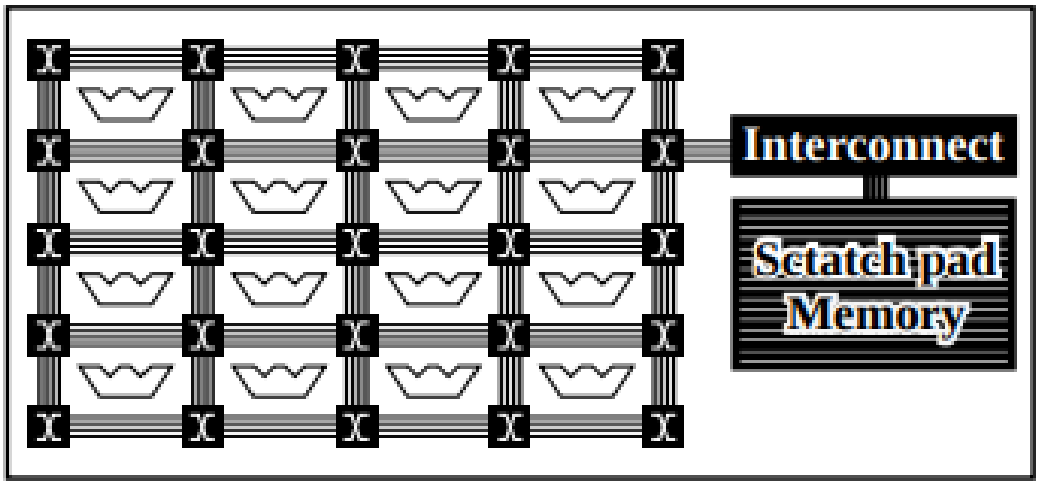

図 2.5　CGRA[4]

いても, スケールアウトによる性能向上を目指すが, サーバサイドを想定しているDPU におけるスケールアウトは, 広帯域幅のメモリインターフェースと Network on Chip (NoC) を使用しておりエッジデバイスには適さない. SIGMA[5] では不規則な形をした疎行列同士の積を高速に行えるように設計されている. 図 2.6 左のように通常の CGRA ではデータ供給スループットが制限されるのに対して, SIGMA では図 2.6 右のように Broadcast してデータを供給しており, 疎行列でもデータ供給スループットを落とさずに提供できているが, 物理的な Broadcast は配線混雑を招き面積効率低下を招く. また, 複雑なマッピングの影響で乗算結果を集めるための加算回路を CGRA の外に用意する必要がある.

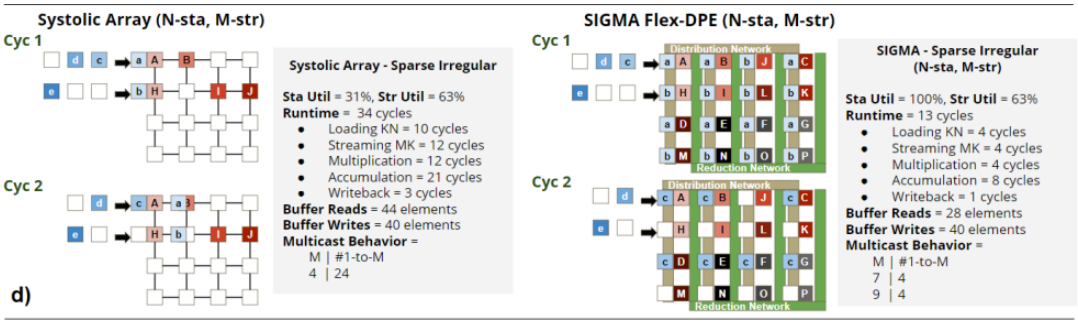
Systolic Array (N-sta, M-str)
Cyc 1
Cyc 2
d)
Systolic Array - Sparse Irregular
Sta Util = 31%, Str Util = 63%
Runtime = 34 cycles
Loading KN = 10 cycles
Streaming MK = 12 cycles
Multiplication = 12 cycles
Accumulation = 21 cycles
Writeback = 3 cycles
Buffer Reads = 44 elements
Buffer Writes = 40 elements
Multicast Behavior =
M | #1-to-M
4 | 24
SIGMA Flex-DPE (N-sta, M-str)
Cyc 1
Cyc 2
Distribution Network
Reduction Network
SIGMA - Sparse Irregular
(N-sta, M-str)
Sta Util = 100%, Str Util = 63%
Runtime = 13 cycles
Loading KN = 4 cycles
Streaming MK = 4 cycles
Multiplication = 4 cycles
Accumulation = 8 cycles
Writeback = 1 cycles
Buffer Reads = 28 elements
Buffer Writes = 40 elements
Multicast Behavior =
M | #1-to-M
7 | 4
9 | 4

図 2.6　SIGMA[5]

第 3 章　先行研究と予備評価

前章では現在までに提案されてきたシストリックアレイの特徴について述べ，疎行列演算との親和性が高いことを強調した．しかし，従来型シストリックアレイにはアドレス計算機構を持ち合わせておらず，疎行列演算には適していない．そこで，アドレス計算機構をもち, なおかつ従来型シストリックよりも面積効率や汎用性を改善した IMAX に説明する．

3.1　先行研究

本節では我々が提案しているアクセラレータである IMAX の基本構造とメモリ参照について説明する．図 3.1 は IMAX の基本 Unit を示したものである．IMAX は 1 列 x64Unit 配置する物理構造を有し，プログラミング時のハードウェアモデルは, 4 列 x64Unit 構成の CGRA である．各 Unit は図3.1 の左側にある EXE/FPU にあるように 3 段パイプラインの 3 入力単精度浮動小数点加減乗算器を 2 組備え，64 ビット幅のレジスタを使用して 2 つの演算を同時実行する．また，パイプラインの初段は 8/16/32 ビット単位の各種マルチメディア演算および固定小数点演算，中段は論理演算，後段はシフト演算等を行うことができる．図 3.1 の青線は read-modify 機能を示しており, 単一 Unit で self-accumulate が可能である．self-accumulate が可能になったことにより, 行列積などを計算した際に生じる累積和を Main memory に返却する必要がなく, Unit 内で計算を行うことができる．従来型の CGRA は縦方向への情報伝播だけでなく, 横方向への情報伝播もするために二次元構造をとっている．横方向への情報伝搬を高速に行うために Broad Cast などを実行するため配線混雑, 遅延時間の増加, コンパイル時間の増加を招いている．また, ローカルメモリから多量の読み出しを行うために，同一内容を保持する複数のローカルメモリを配置して複数ポート化する必要がある．つまり，広帯域なバスを持つローカルメモリであれば，一度のロードでアドレスの連続したデータを読み出し可能であるが，離散アドレスに対するデータロードを行うためにはメモリのポートが複数必要である．しかし,IMAX は従来型 CGRA と異なり, マルチスレッディングを導入したリニアアレイ構造を取っている．ここで, 列マルチスレッディ

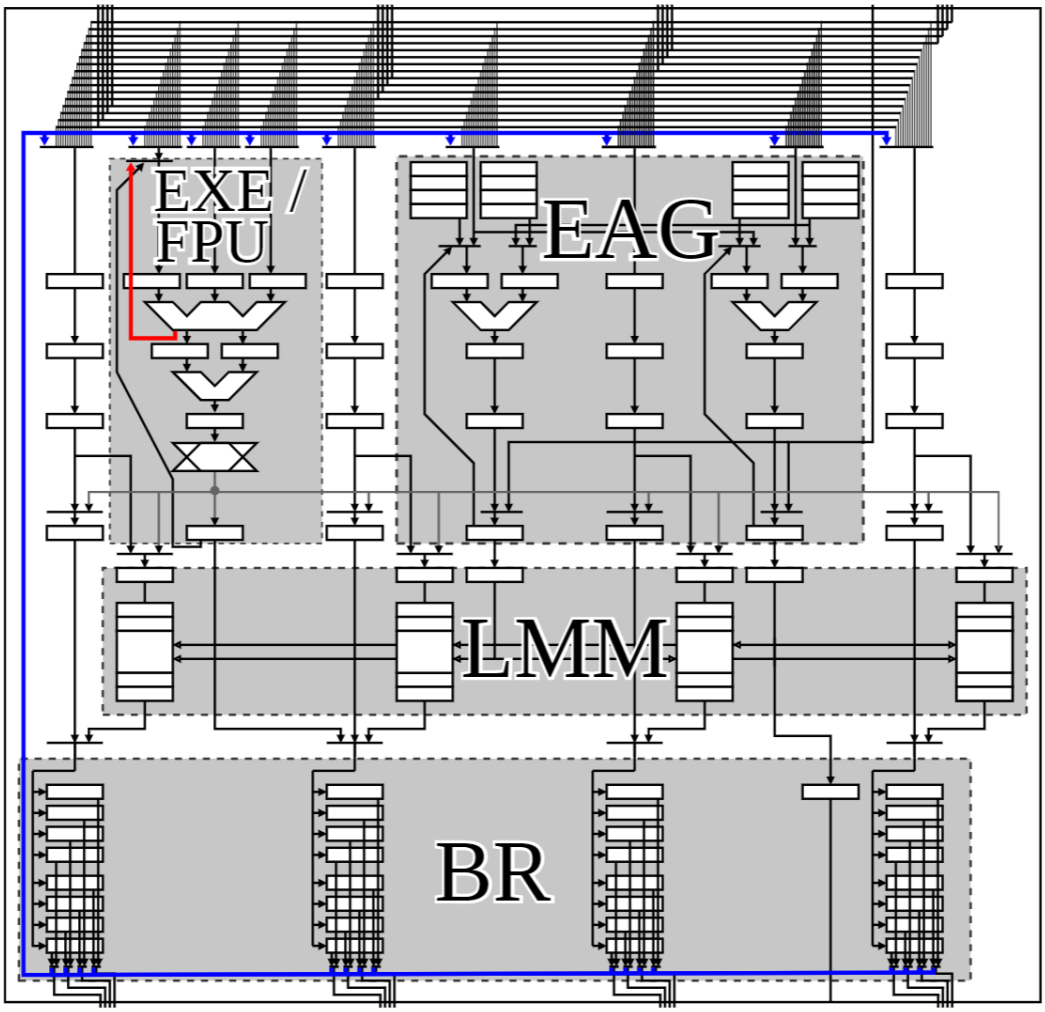

図 3.1　IMAX 概要

ング導入前の一般的なユニット構成を図 3.2(a)，図 3.2(b) にマルチスレディング導入後の IMAX を示す．図 3.2(b) のように 1 つのユニットが 4 サイクルを使用して，論理的にと同じ機能を実現することで改善することができる．演算器を 4 段パイプライン化し，マルチスレッディングを行うことで，1 つの演算ブロックで 4 つの演算ブロックが行っていた演算を行うため，ローカルメモリを備えた演算ブロックを 4 分の 1 に削減することができる．図3.2(a) では 1 つのユニットにおける演算

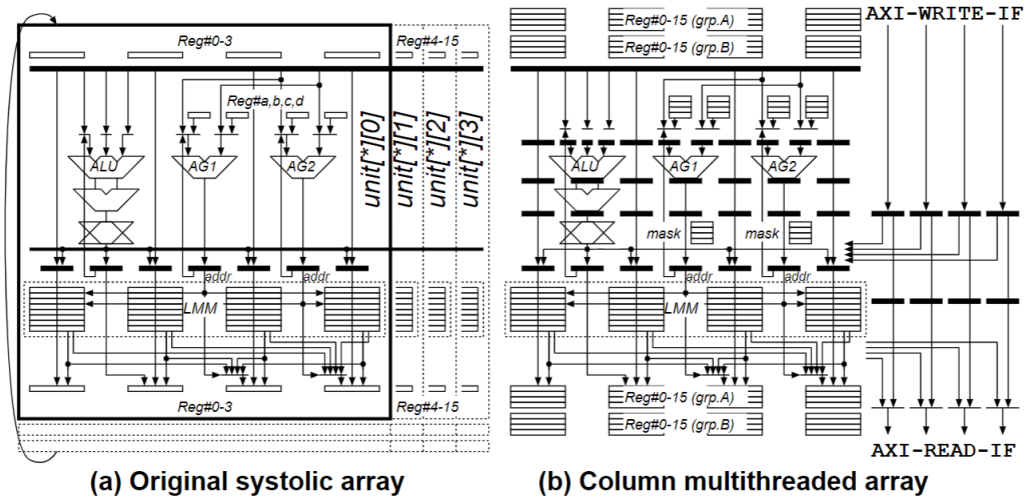

図 3.2　マルチスレディングの概略

遅延が 2 サイクルであるのに対して，図 3.2(b) では 8 サイクルである．ここで, 先ほどの IMAX 概略図図 3.1 における赤線について説明する．EMAX6 ではユニットの 4way マルチスレッド化にあわせて演算器を 4 段パイプライン化しており，ループカウンタのデクリメントにそのままでは 4 サイクルかかる．しかし，デクリメント自体は 1 サイクルにて実行できるため，4 サイクルを使用すれば 4 列分の依存関係のあるデクリメントが可能である．そこで,LOOP0,LOOP1 をループ制御変数として定義し,LOOP0,LOOP1 をそれぞれループの終了判定に用い，全てが 0 になった際に演算を終了する．そのために, 赤色で示したループ終了判定結果を 1 サイクルで通知するためのパスを追加した．また，ループ先頭であることを示すフラグとして INIT0, INIT1 を実装し，アドレス計算時にこれらを用いてロードする値を選択し，多重ループにおけるアドレス計算を可能とした．3 重ループ一括実行を実装することも可能であるが，アドレス参照範囲が膨大になることが予想され，大容量 LMM が必要となり面積の増大に繋がることから，3 重ループ目はマルチチップ構成に対応付ける．

次に,IMAX の Memory Map とメモリ転送の概略を図 3.3 に示す．IMAX に関連する物理メモリ空間は，(1) CPU 物理メモリ空間と LMM の間の DMA を制御する FPDDMA-CH1 DMA 制御レジスタ空間（DMA control register space:4KB),

（2）LMM を CPU 物理メモリ空間に写像して DMA により参照する LMM DMA 空間（pAddr2 space:2GB）および IMAX 内のレジスタを PIO により参照する制御レジスタ空間（pAddr2+ space:2GB），（3）LMM との DMA が可能な DDR メモリ空間（pAddr space:2GB）から構成される．このうち，IMAX との物理的接続により参照できる空間は，pAddr2 space:2GB および pAddr2+space:2GB である．IMAX はベアメタルではなく,Linux の仮想メモリ空間上で動作するため, ユーザーは直接物理アドレスを参照できず,mmap で割り当てられた仮想アドレスのみを参照する.(1) にある DMA コントロールレジスタを制御することにより,(2) にある内容を IMAX へ DMA 転送することができる.IMAX の各 Unit は DMA 転送されてきたデータを自分のローカルメモリに取り込むかどうかを判断するために事前にユーザから PIO 経由で送られたアドレス範囲の情報を使用する．しかし, ユーザーから送られるアドレス範囲情報は仮想アドレスなのに対して,DMA から転送されるアドレスは物理アドレスなので比較できない．そこで,IMAX 全体を制御する FSM が DMA から転送されたデータの物理アドレスを仮想アドレスに変換して互いに比較できるようにする．CPU 側が config 情報を設定し, 各 Unit が DMA 転送で送られた値を各 Unit が持つアドレス範囲情報と比較し, 該当する場合ののみ自律的に取り込む仕組みにより,IMAX 内で複雑な DMA コントローラを持つ必要がなくなり, 複数枚の CHIP で動作する際に回路面積削減につながる.

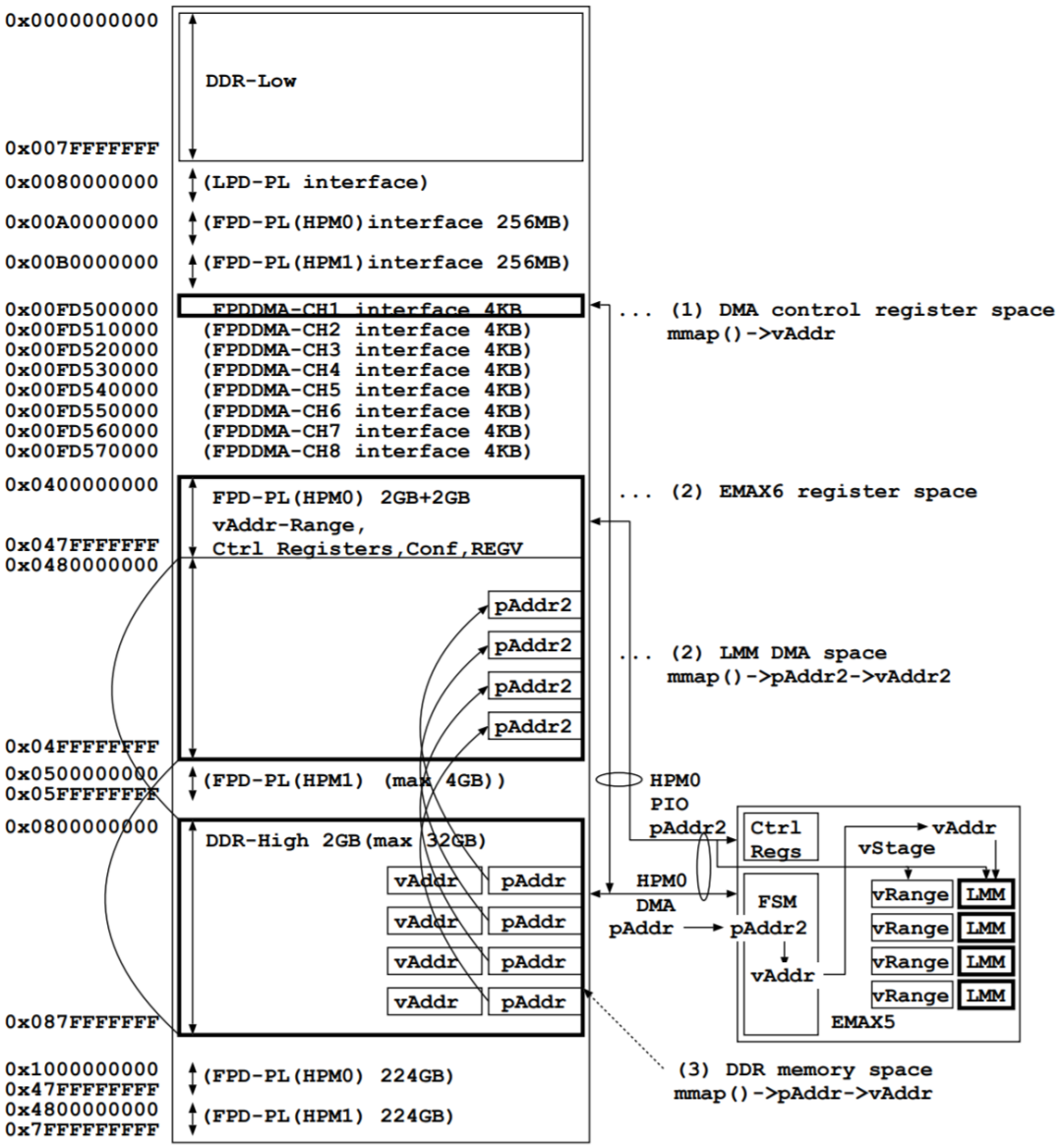

図 3.3　MemoryMap の概略

3.2 予備評価

予備評価として, 密行列どうしの行列積を IMAX を用いて行う．行列積は古くから計算機での高速化が盛んに研究されているアプリケーションである．古典的なニューラルネットモデルであるパーセプトロンや Deep Learning の全結合層も行列積で実装される．図 3.4(a) に mm のデータ構造，図 3.4(b) に C 言語による一般的な実装，図 3.4(c) に各 unit に積和演算器を 1 つ備える H+1 (高さ)×W (幅) 構成の IMAX の同一時刻におけるスナップショットを示す．出力 C[row][col] に対応する A[row][*] × B[*][col] の積和演算を H 組ごとに分割し,1 列の unit[*mod H][col mod W] に対応させてパイプライン実行すると，H×W の全演算器を利用でき効率がよい．例えば左端列では，結果 C0c, C08, C04, C00 に必要な乗算 A00 × B0c，A01 × B18, A02 × B24, A03 × B30 を同時刻に行い，最終段から C00, C04, C08, C0c, ..., C0(M-4) の部分和を毎サイクル出力しローカルメモリに累算する．すなわち，H×W の演算スループットにより，毎サイクル W 個の部分和を出力する．これを M/H 回繰り返すことにより，完全な C00, C01, ..., C0(M-1) が得られる．図 3.4(d) はさらに，配列 A を N チップに行分割して並列処理し，各ローカルメモリが，配列 A の 1 行（M 要素）× GRP 行分と，配列 B の 1 行（M 要素）を収容できる場合の実装である．矩形内の 5 重ループが全 IMAX チップを 1 回起動する処理に対応し, chip ループの各イタレーションが, 各チップにおける配列 A の GRP 行分と配列 B 全体の行列積に対応する．外側ループの blk が 0 から M-1 に増加する間, 各チップの最終段を除くローカルメモリは配列 A を GRP 行分保持し, 最終段のローカルメモリは $\mathrm{C}\times 0, \mathrm{C}\times 1, \ldots, \mathrm{C}\times(\mathrm{M}-1)$ を GRP 行分保持しつつ更新する．一方, 配列 B は, blk を更新する度に対応する H 行分を全チップにブロードキャストする必要がある．以上の演算に要する理論的サイクル数は, IMAX 起動時の遅延 (H+1 サイクル) およびローカルメモリの入れ換え時間を除くと $\mathrm{M}^3/(\mathrm{H}\times\mathrm{W}\times\mathrm{N})$ となる．また, ローカルメモリが $\mathrm{A},\mathrm{B},\mathrm{C}$ 全体を収容できる場合の理論的 DDR とローカルメモリ間転送量が $\mathrm{M}^2\times 3$ であるのに対し, 各ローカルメモリが A と C を $\mathrm{M}\times\mathrm{GRP}, \mathrm{B}$ を $\mathrm{M}\times\mathrm{H}$ のみ収容できる場合の転送量は, A と C が各々 M^2, B が $\mathrm{M}^2\times\mathrm{M}/\mathrm{GRP}$ ($\mathrm{M}=\mathrm{GRP}$ の場合 M^2 に一致．N に非依存．ブロードキャスト前提) と最適になる．ここで IMAX の動作一覧を表 3.1 に示した. 表 3.1 に基づく実行の流れは以下の通りである．まず,DRAIN によって

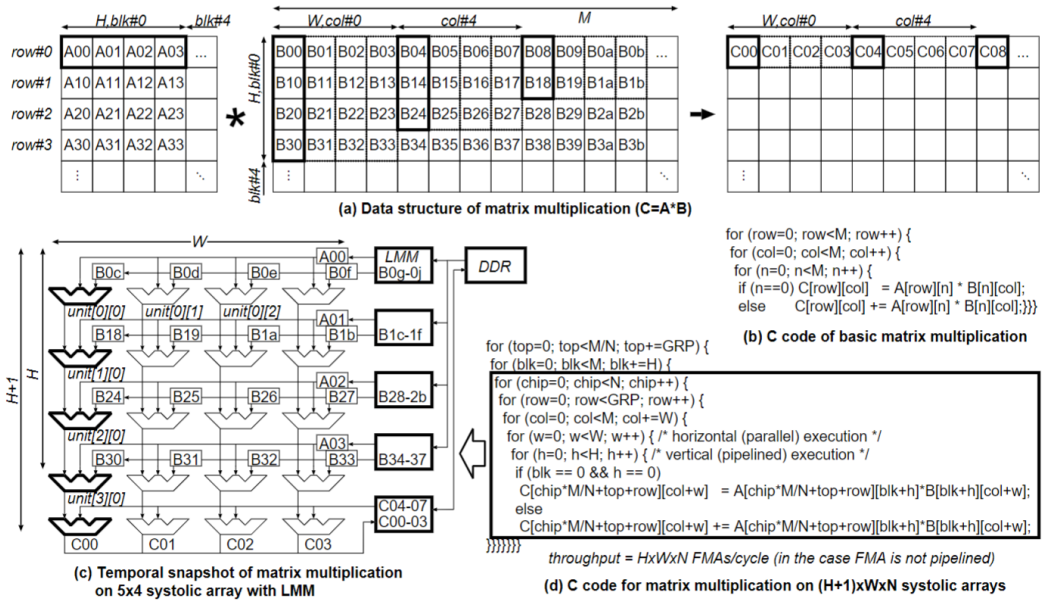

図 3.4　行列積の実装

前回実行時のローカルメモリの内容を読み込み，CONF によって各ユニットへ命令をマップし，REGV でレジスタを初期化する．次に,RANGE によって有効となるローカルメモリを設定し，演算に必要なデータを DMA(LOAD) でローカルメモリへ転送し，EXEC で演算を実行する. ここで, 各動作の実行時間の内訳を調べるために正方行列積を実行した結果を図 3.5 に示した. ただし, 使用した CHIP 数を 1 とする．実行結果により,LOAD と DRAIN がボトルネックとなっていることが分かった．実行時間を削減するには次のような手段が考えられる．

1.Local Memory の利用率向上および次回実行時の再利用率向上させる.

2. 疎行列性を活用し,LOAD の絶対量を削減する.

本研究でこの二つの削減方法について取り組み, 行列積の実行時間削減を目指す.

表 3.1　IMAX 動作シーケンス ステート一覧

ステート名	動作の説明	PIO/DMA
CONF	命令マッピング	PIO
REGV	レジスタ情報設定	PIO
RANGE	ローカルメモリ情報設定	PIO
DRAIN	DMA 転送 (ローカルメモリ-主記憶)	DMA
LOAD	DMA 転送 (主記憶-ローカルメモリ)	DMA
EXEC	演算実行	PIO

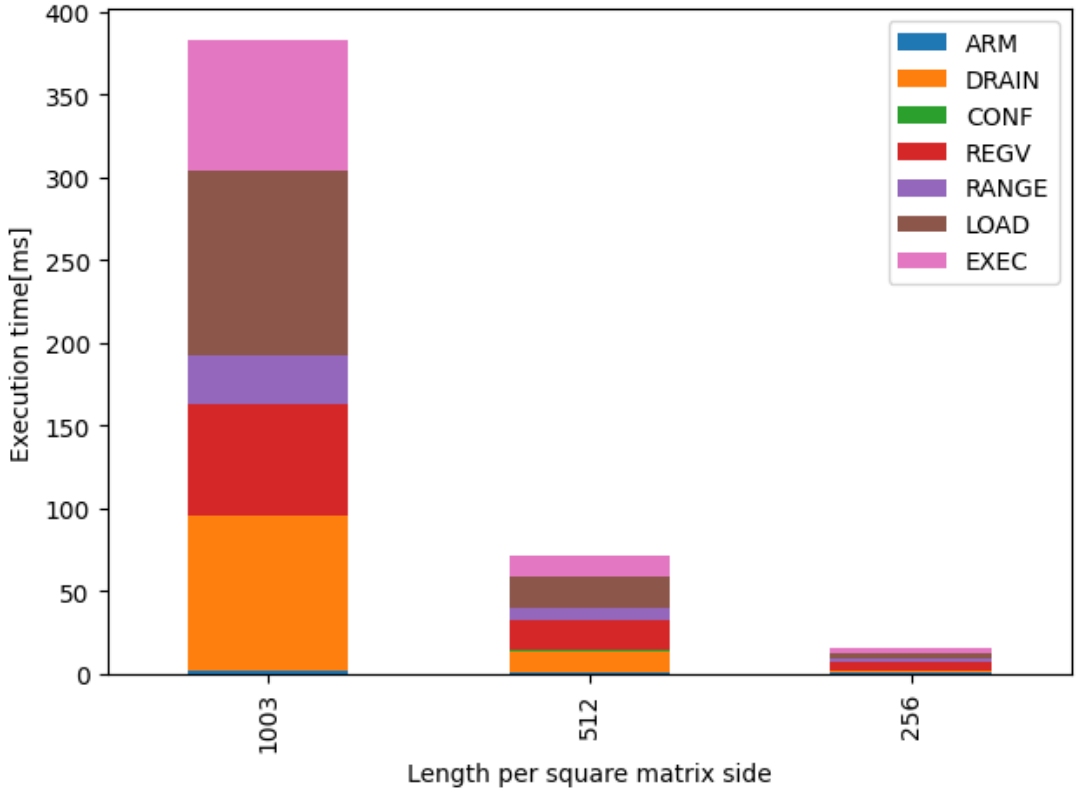

図 3.5　各行列サイズでの実行時間

第 4 章　提案手法

以前の章では,IMAX が面積効率やアーキテクチャの優位性を示した．しかし,IMAX には二つの問題が存在している. 一つに, パラメータのチューニングの難易度が高く, ハードウェアに詳しくないプログラマーにとって使用が困難だったこと．二つ目に疎行列積に対応しておらず,DNN や SLAM などのアプリケーションで疎行列性を活かした計算できないことが挙げられる．この章では疎行列のスケジューリング方法を提案する. まず, 従来とは異なる疎行列形式に格納し, 二通りの演算フローを示す．次に実験のためにチューニングが必要なパラメータを洗い出し, 定式化する．最後に定式化の制約内でパラメータを探索することで, 最適なパラメータを発見方法を提案する.

4.1　独自スパース フォーマット

本節では,IMAX で疎行列積を行うためのフォーマットを提案する．疎行列演算を行う際は基本的にゼロ部分を取り除き, 非ゼロ部分のみを特定のフォーマットに格納する．代表的なフォーマットとして COO（Coordinate list）や CSR（Compressed Sparse Row）や CSC（Compressed Sparse Column）などが挙げられる. また CSR を発展させたものとして,Jagged Diagonal Storage (JDS) が挙げられる．本研究では CSR と JDS を IMAX 用に定義し直したものを使用する．ここで,CSR の例を図 4.1 に示した.CSR は行列の非ゼロ部分を val に格納する他に, col に非ゼロ部分がどの行に格納されているかの情報格納し,ptr にその列での非ゼロ個数を格納する．行列の非ゼロ部分に偏りがない場合は CSR が適しているのに対して, 偏りがある場合は行列をブロッキングできず, 疎行列演算の性能が著しく低下する恐れがある. それに対応するために JDS が提案されている.

JDS は列の長い順番に並べ替えて, 行方向に格納する方式である．ここで,IMAX のためにいくつかの変更を加えた JDS の実装手順を図 4.2 に示す．図 4.2.a は初期状態の行列を示していて, 行列格納方向は問わない．図 4.2.b は CSR のように左に非ゼロ要素を詰めて格納し, 要素が本来入っている場所の行情報を index に格納する．格納するための計算量は $O(N^2)$ である．次に図 4.2.c のように行方向の長さに

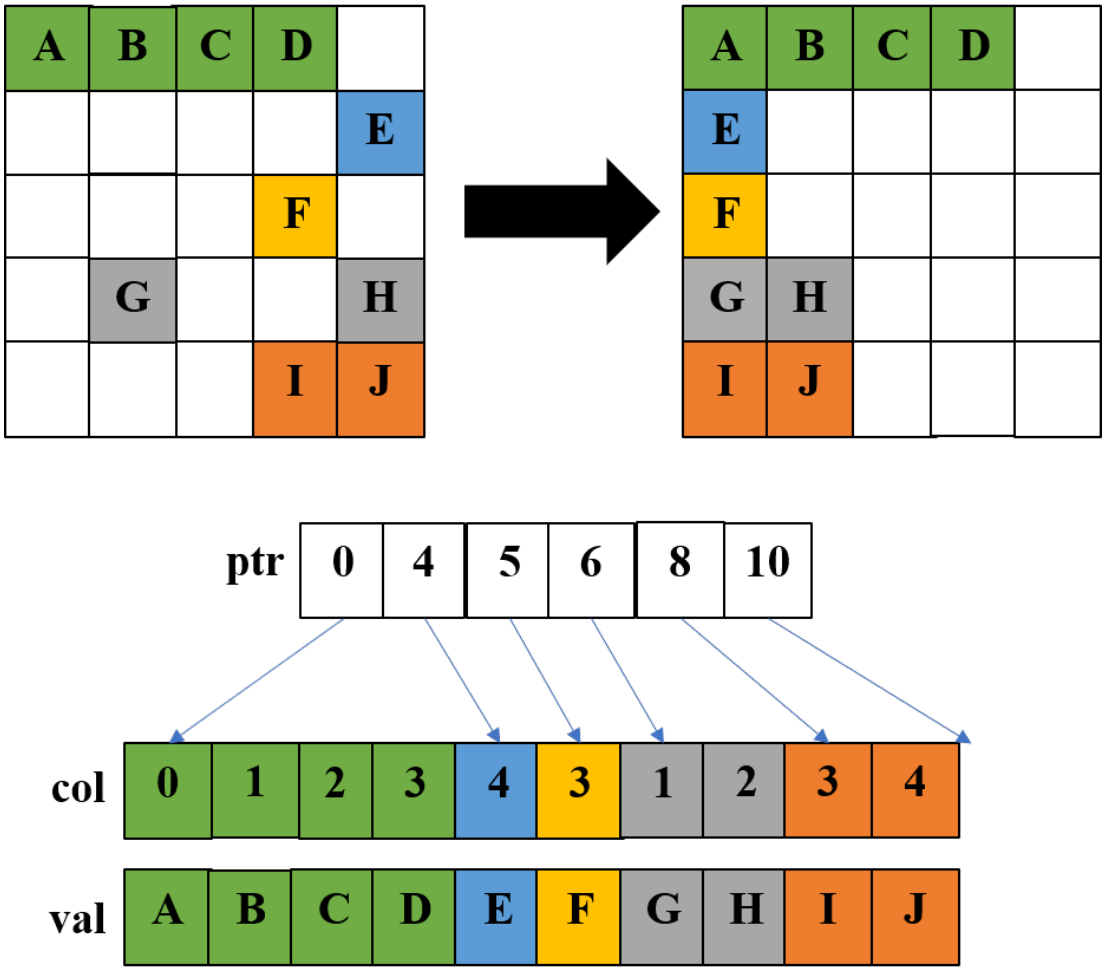

図 4.1　CSR の例

基づいて並べ替えを行う. 図はわかりやすくするために値も並べ替えているが, 実際は行ごとの非ゼロ数を降順に並び替えるだけで, 行列の中身自体は並び替えない. ここで使用したソートの方法について説明する. 分布数え上げソートは値をキーにして, キーの出現回数とその累積度数分布を計算して利用することで整列を行うアルゴリズムで並べ替え範囲が分かっていて, 要素数が多くない状況下では O(N) で計算できる. 分布数えソートは今回の問題設定においてクイックソートよりも優れているので選択した. 最後に図 4.2.d で実際に値の並び替えおよび配列への格納を行う. IMAX で計算をする際, スケジューリングの都合上, 列方向に値を格納する.

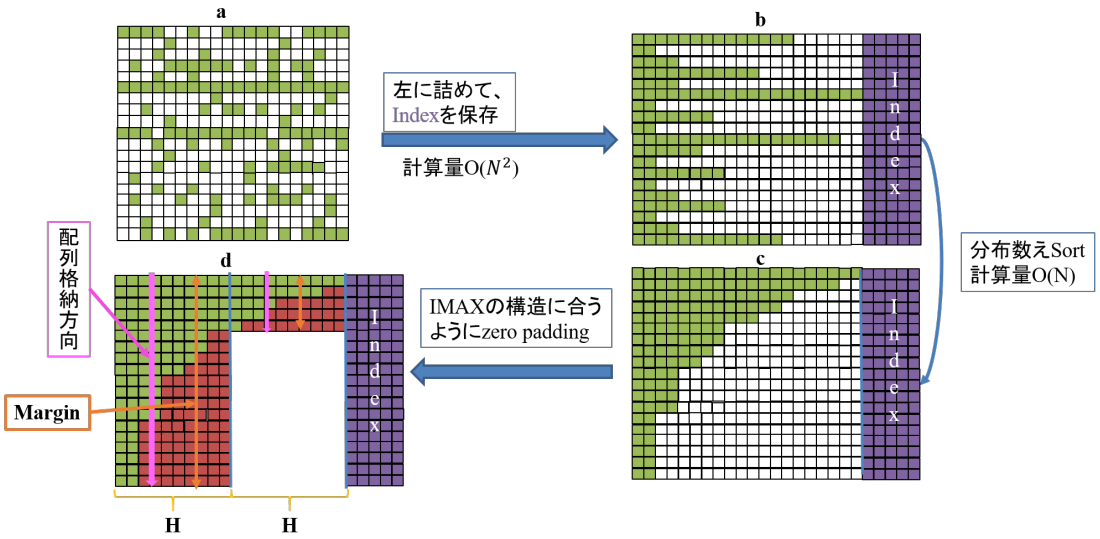

図 4.2　JDS の例

赤色に塗られている部分はゼロで埋めている部分 (Padding) でブロック単位で分割しやすくしている．図 4.2.d の H は IMAX の高さを示しており,Margin は H ごとにどの深さまで列を確保すべきを示している．Margin を扱うことによって余分な Padding を IMAX へ転送することを防いでいる．

CPU や GPU などの疎行列演算では値と行の位置情報が入っている index を分離するのが普通である．しかし,IMAX の場合はアドレス読み出し機構を備えており, 値と index をセットにして格納する方法も考えることができる．格納パターンの違いを図 4.3 に示した. 図 4.3.a では値と index を別々に 32bit 単位で格納しているのに対して, 図 4.3.b は index 部分を削減し, 値と index を 64bit 単位で管理している. 絶対的な格納量は変わらないものの, フォーマットを行う際に書き込み範囲が図 4.3.a に比べて小さくなるため, フォーマットの時間を削減できる可能性がある．次節では図 4.3.a と図 4.3.b を用いた実際のスケジューリング方法を提案し, 議論する．

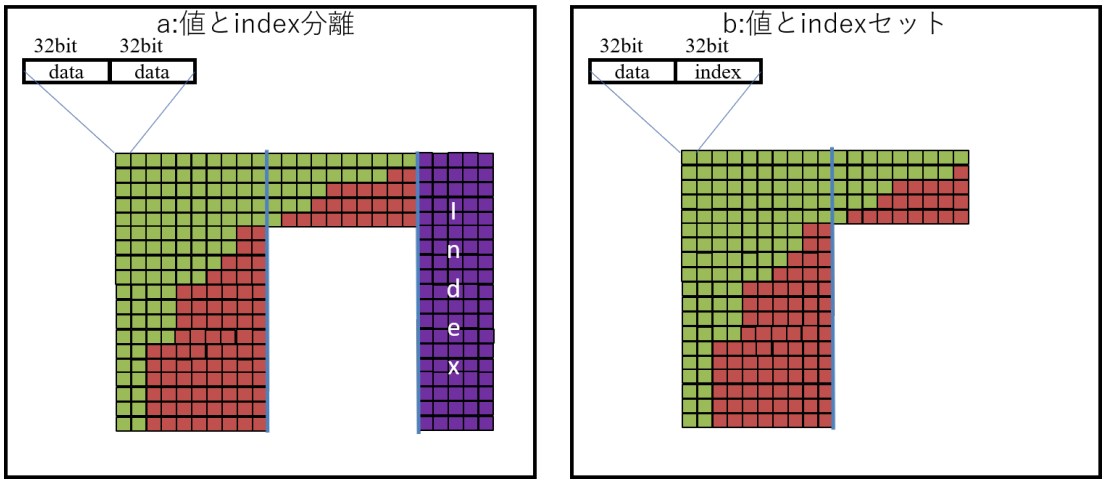

図 4.3　値と index の格納パターン

4.2　IMAX の疎行列積スケジューリング

本節では前節で説明した二種類の疎行列フォーマットに適した疎行列積スケジューリングの方法を示す．ただし, 掛けられる側を疎行列 A とし, 掛ける側を密行列 B とする．また IMAX 内の各 Unit が持つ Local Memory を以後 LMM とする．まず Format において値と index を分けて考える従来型の場合について検討する．スケジューリングの概要を図 4.4 に示した．スケジューリングの基本方針は以下の通りである．まず Unit を 2 つ使用して疎行列 A に対応する密行列 B の offset を取得する．1 回あたり IMAX の横幅 W×Port×2(32bit×2) の offset を取得でき, 残りの Unit でその offset を用いて密行列 B の該当箇所を Load する．本研究では横幅 W を 4 とし Port 数を 2 としたため,16Unit 分の offset を一度に取得することができる．この 16Unit を 1 セットと考え,IMAX 最終段まで同様の演算を繰り返す．以下図 4.4 の簡略な説明を行う．Unit1 から aindex と aindex1 を呼び出している. これは JDS で格納した index に相当する．aindex 一つで二カ所の index 情報を格納しており,64bit 単位である．aindex には疎行列の非ゼロ部分の行情報が入っているが, これは掛ける側の密行列 B の列情報に対応する．このことは通常の密行列どうしの行列積と同様のことである．Unit2 では Unit1 から伝搬してきた aindex,aindex1(密行列 B の列情報) と IMAX を横断して伝播する密

行列 B の行情報 boffset を足し合わせて LMM 内にある密行列 B の値を Load するために使う boffset1 ,boffset2 を取得する．Unit3 では Unit2 から伝搬してきた boffset1,boffset2 と Base を足し合わせてアドレスを取得し, そのアドレスを用いて密行列 B の値を Load する．疎行列 A の値も同様に Arow と Base1 を足し合わせたアドレスを用いて Load する．B は Unit 間で共通なので Base は同じもの用いるが,A の Base は Unit ごとに異なっている．Unit4 では Unit3 から伝搬した a1 と b1 のペアを乗算し, 下段へ伝搬する．先ほど説明した通り,boffset1 は二カ所の offset を含んでいるので,32bit シフトしてもう片方の offset を Unit5 で用いる．Unit5 では Unit4 から伝搬した c1 と a2 と b2 の乗算結果を足し合わせて下段へ伝播する．以下同様の計算を行うことで疎行列積の積和部分を実現している．以降, この手法を Index-Value-Separate 法 (IVS 法) とする．

次に Format において値と index をまとめて考える方式について検討する．スケジューリングの概要を図 4.5 に示した．スケジューリングの基本方針は以下の通りである．まず Unit を 1 つ使用して疎行列 A の要素を呼び出す．呼び出された疎行列 A(64bit) のうち 32bit は値で 32bit は index である．次の Unit で前 Unit から伝搬した A のうち index 部分を用いて密行列 B のアドレスを計算し, 密行列 B を Load する．次の Unit は 2 段前の Unit から伝搬した A のうち値部分を用いて前段の Unit から伝搬した密行列 B との乗算を行う．つまり 3Unit で乗算一つを行うことができる. このサイクルを 1Unit ずつ下段へシフトしながら行うことで, 図 4.4 のように index 計算のために Unit を割かなくてよい．また次節で述べるように,B の行方向へのシフトは for ループ連続実行の外で行う．以下図 4.5 の簡略な説明を行う．Unit1 では a1 を Load する．Unit2 では前段から伝搬した a1 の index 部分を用いて b1 を Load し,a2 も Load する．Unit3 では Unit1 から伝搬した a1 の値を用いて,Unit2 から伝播した b1 と乗算し, 下段へ伝播する．以下のサイクルを最終段まで繰り返す．以降, この手法を Index-Value-Join 法 (IVJ 法) とする．

密行列どうしの乗算で 60Unit を乗算に使えるとした場合, 図 4.4 の場合は 48(16×3)Unit を積和に使えるのに対して, 図 4.5 は読み出しのみを行う最初の 2Unit を除いた 58Unit を使うことができる．これらの結果から,IVJ 法の方が優れていることがわかった．したがって, これ以降は IVJ 法を用いて提案および実験を行うものとする．IVJ 法の問題点としては,32bit 単位で値と index を格納している

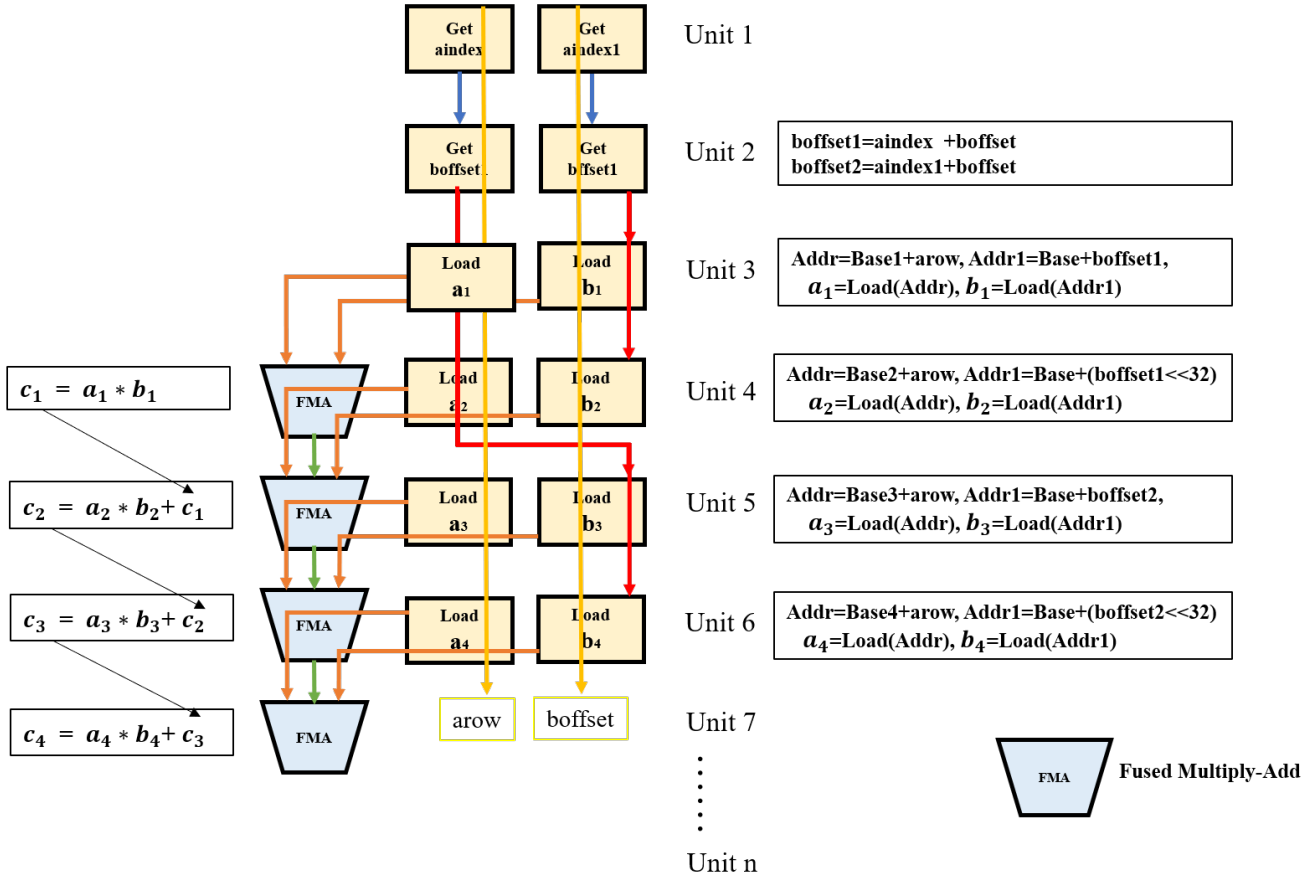

図 4.4　IVS 法の概要

ため,IMAX が備えている SIMD 演算機構を活用できないことである．SIMD 化を実現するには, 密行列 B を 32bit×2 で読み出したときに疎行列 A の 32bit を index 部分にコピーして 32bit×2 に拡張し, 密行列 B の二要素と計算する機構が必要であるが IMAX にはそのような機能を備えていない．そこで, 図 4.6 のように FMA の入口まわりを変更した．Address Calculator や Local Memory や FPU の構成はそのままで,FMA に入る直前に 64bit のうち 32bit をもう片方にコピーする機能を回路に追加した．足りない 32bit をホストから転送するのではなく IMAX 内でコピーして用いることができるので, メモリーの転送量削減にもつながり効率の良い

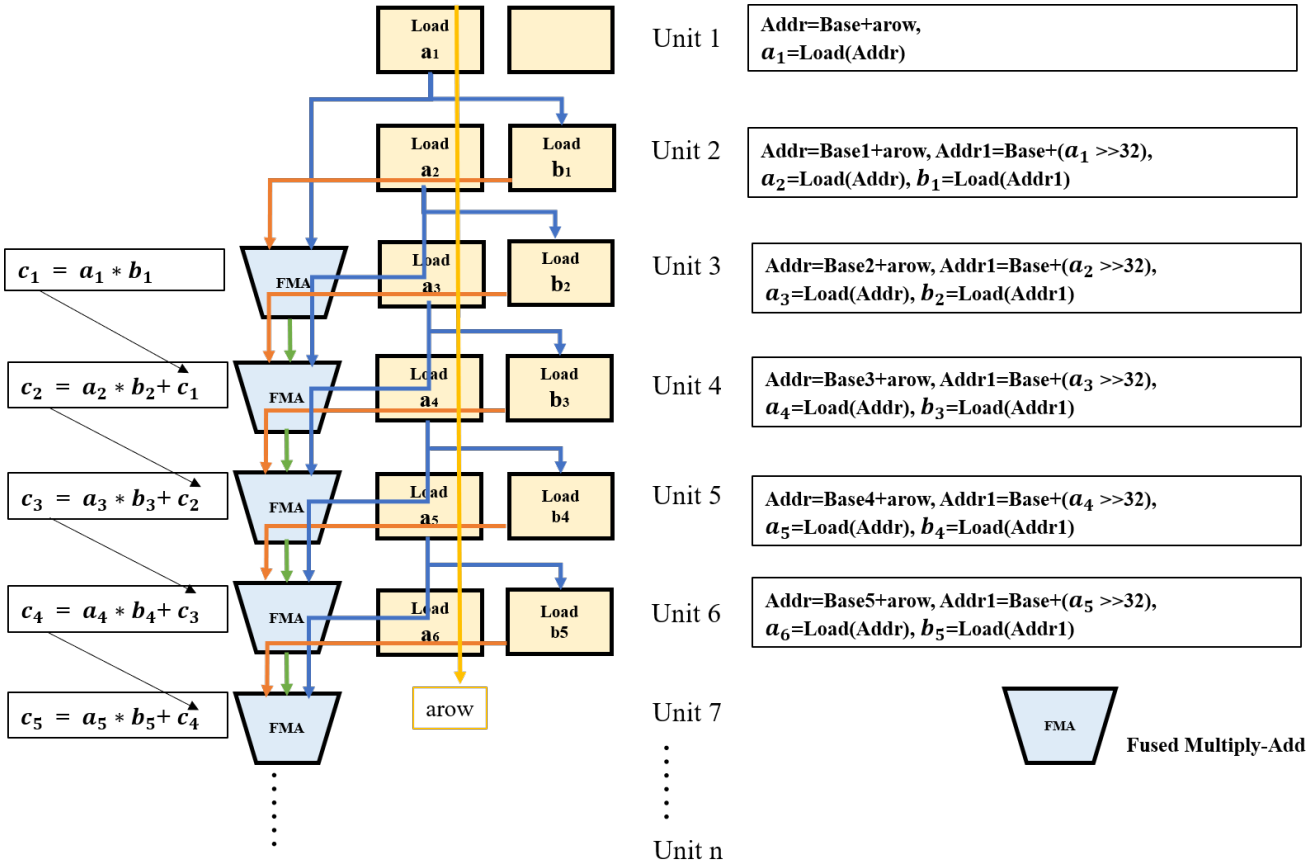

図 4.5　IVJ 法の概要

演算を可能とした．これらの変更により疎行列 A と密行列 B の SIMD 演算が可能となった．この機能は疎行列だけでなく, 様々なアプリケーションにも使用することが可能である．つぎに最終段における演算結果を密行列 C に保存する方法を示す．JDS では疎行列 A を並べ替えるため, 演算結果の密行列 C を元に並べなおす必要がある．従来の CGRA であればアドレス計算機構を備えておらず, ホスト側で並べなおす必要があるが IMAX はアドレス計算機構を備えているので, IMAX 内で並べなおすのではなく, 適切な場所に直接書き込むことができる．ここで, 最終 Unit の概要を図 4.7 に示した．並べ替え先の行情報を格納した perm_offset を初めの Unit から伝搬し,perm_offset と Base を足して得られるアドレスを演算結果保存先とした．赤矢印は先行研究で説明したように self accumulate 機能で, パイプラ

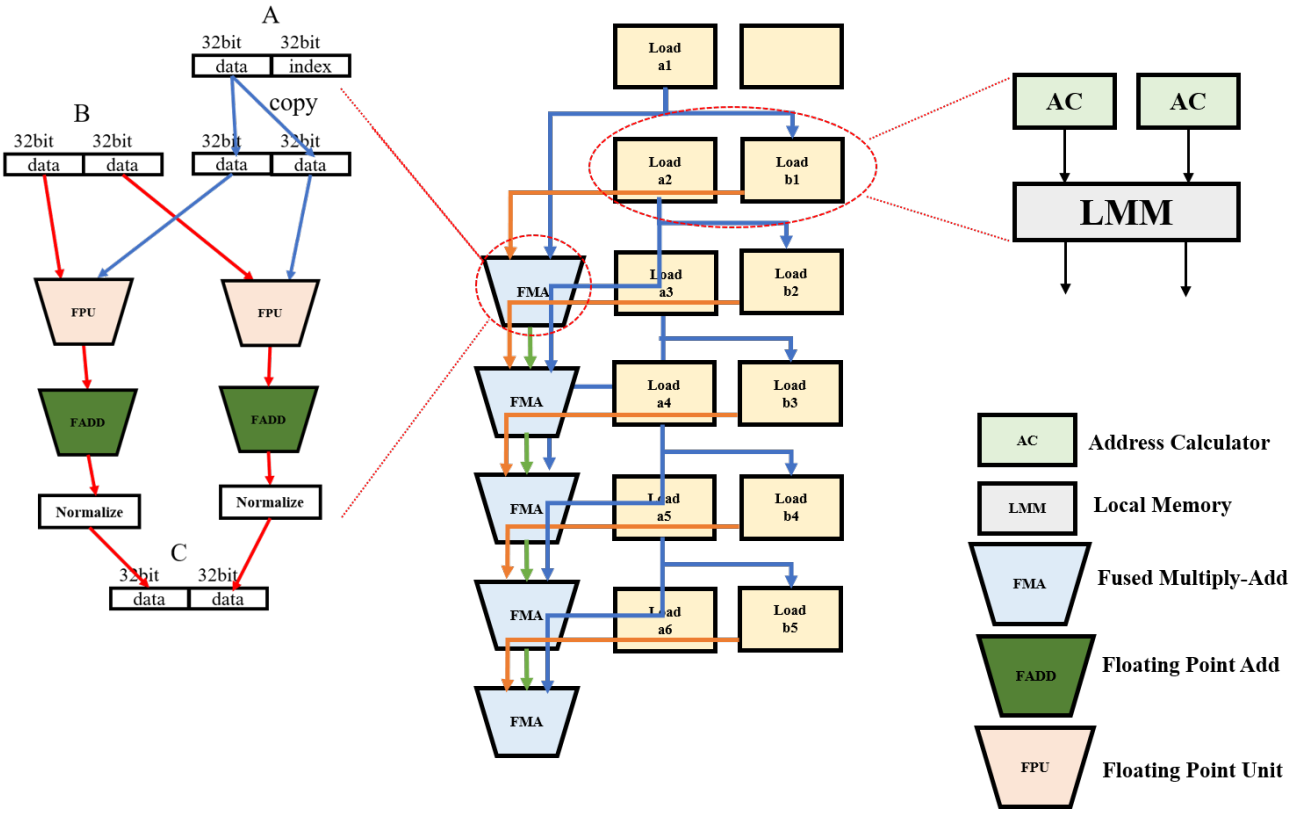

図 4.6　機能追加の概要

インで伝搬され続ける値を足し続けることができる.

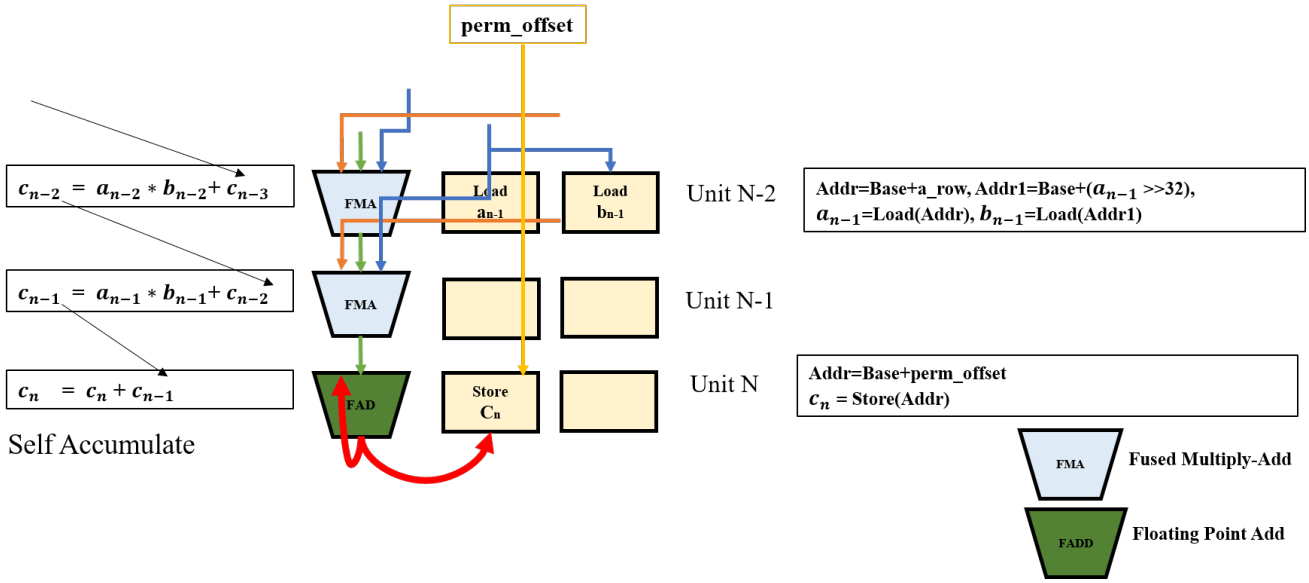

図 4.7　最終段の概要

4.3　Local Memory 利用率改善

先行研究で説明したように IMAX で行列計算を行う場合, 疎行列 A との乗算に使われる密行列 B を各 Unit へグループ化して格納していた．これは行列を分割して小分けで計算することでキャッシュ利用率をあげるブロッキング手法と考え方は同じである．密行列 B に対して, 疎行列 A は各 Unit の Local Mermoy(LMM) に行列一辺分のみ格納していた．疎行列 A も密行列 B のようにグループ化して LMM 内に値を確保すれば, メモリ利用率をあげることができる．しかし, 疎行列 A の Group 化に必要な制御変数を IMAX 内のループ制御レジスタに割り当てると, IMAX が連続実行する際に扱うことができるループ制御レジスタの上限を超えてしまうため, 疎行列 A のグループ化に必要な変数を IMAX 内に割り当てることができない．したがって A をグループ化して LMM 内に確保する際は一度連続実行を中断し,LMM 内の中身はそのままで A グループ制御変数の更新および, それに伴うベースアドレス更新のみを行うフェーズを用意した．これにより連続実行は中断されるものの, 疎行列 A の頻繁な入れ替えを防ぎ, メモリ再利用率を向上させることができる．ここで図 4.8.a に疎行列 A をグループ化する前のメモリ割り当て

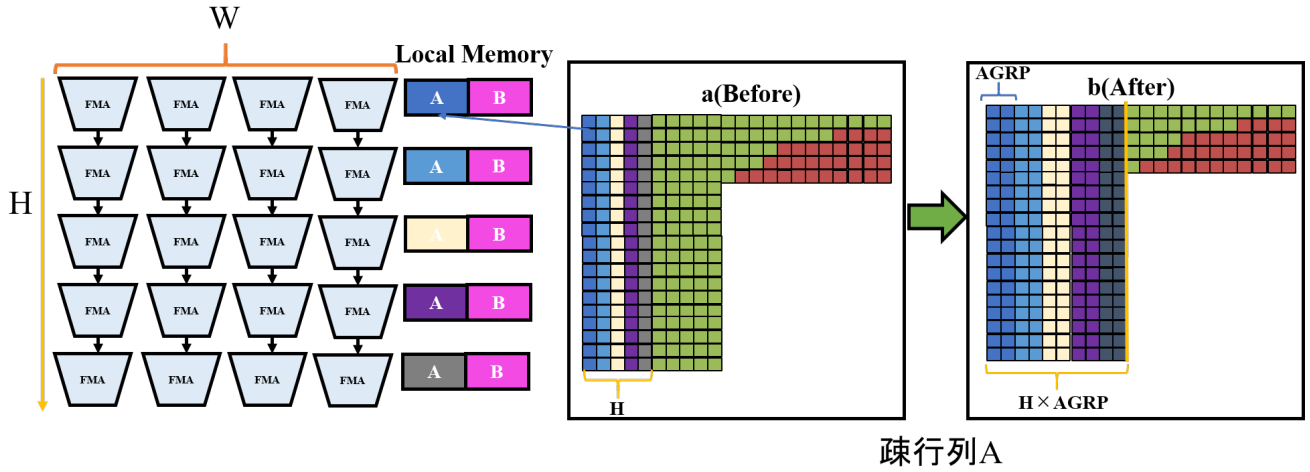

図 4.8　AGRP の概要

概要を示し, 図 4.8.b に疎行列グループ化後のメモリ割り当て概要を示した.W は IMAX の横幅で,H は高さとし,AGRP は A をグループ化する際に用いる変数とする．AGRP は実行状況に応じて柔軟に変更できる. 行列サイズに応じて変更することができるだけではなく,LMM へ疎行列 A の一部を転送する度に変更できる．これにより, 序盤は LMM の容量を超える手前まで AGRP を増やし, 終盤の計算では疎行列 A の横幅自体をはみ出ないように減らすといったスケジューリングが可能である．図 4.8.a では一回あたり A の高さ ×H 分しか IMAX 全体で確保できないのに対して, 図 4.8.b では A の高さ ×H×AGRP 分確保できる．従来,LMM に行列がすべて乗りきらない場合は密行列 B を LMM に残して疎行列 A の内容を入れ替えながら計算していたが今回の変更に伴い, 疎行列 A も最大まで LMM を活用できるようになったため密行列 B を入れ替える方に選ぶ. 密行列 B は図 4.8 でみてとれるように全 Unit に同じ内容を送るようなスケジューリングをしている. 見かけの転送量は疎行列 A と密行列 B とで LMM を 2 分割しているので同じであるが, 先行研究で説明したように IMAX は各 Unit が自律的に DMA 転送された内容を取り込むようになっており, 密行列 B の内容を一度送るだけで各 Unit が自律的に取り込むため, 複数回同じ内容を送る必要がなく, 転送量削減に繋がる.

4.4 Local Memory 最適化

本節では疎行列積の際に用いる IMAX のパラメータを定式化して, 最適なパラメータを選ぶ実装方法を示す. 制約条件を式 (4.4.1) から式 (4.4.5) に示した. ただし, 入力行列をそれぞれ疎行列 A, 密行列 B とし演算結果の行列を密行列 C とする. Acolumn, Arow をそれぞれ疎行列 A の列と行をとし,Bcolumn, Brow をそれぞれ密行列 B の列と行とした. H は予備評価の際に説明したように IMAX が使える高さであり,W は横幅である. NCHIP は IMAX の CHIP 数を表している. LMMSIZE は LMM の容量を表しており, 本研究では 64kbyte とした. また, 結果を保存する最終 Unit の LMM は密行列 C が 64Kbyte 使用し, そのほかの Unit の LMM は疎行列 A と密行列 B が 32Kbyte ずつ使用するものとする. AGRP と BGRP と CGRP はそれぞれ疎行列 A, 密行列 B, 密行列 C の LMM に応じて調節できるパラメータである. 今回は AGRP,BGRP,CGRP を調節することで LMM の利用率を上げる. IMAX は H 単位で実行が行われ, 今回の手法では Acolumn に沿って H 分の積和が実行される. ここで IMAX パラメータの割り当てを可視化した図を図 4.9 に示した. 次に,IMAX パラメータの制約式 (4.4.1) から式 (4.4.10) の説明をそれぞれ行う. H 単位で実行して割り切れない場合へ対応するために式 (4.4.1) を用意した. Acolumn が H で割り切れない場合, その余剰分の Arow を 0 で埋めるものとする. これにより行列サイズの制限を無くし, 長方形行列も計算可能になった. IMAX は H 単位で実行が行われ, 今回の手法では Acolumn に沿って H 分の積和が実行される. IMAX は先行研究で説明したように数段階の For-Loop 連続実行が可能であり, その際に LMM をいかに再利用するかが性能向上の鍵である. 本研究では LMM の使用率を最大限高めるような実装をした. 式 (4.4.2) から式 (4.4.3) は疎行列 A の制約条件を表している. 式 (4.4.2) の Arow × AGRP は DMA 転送 1 回あたり LMM へ送るサイズである,LMMSIZE の半分である 32Kbyte を超えないように制約している. 制約条件に 4 がかけられているのは byte 単位に変換するためである. 制約条件に 2 が掛けられているのは A を値とアドレスのセットで扱う 8byte 単位で管理するためである. 式 (4.4.3) は Block サイズが疎行列 A 全体のサイズを超えないようにするための制約条件である. 式 (4.4.4) から式 (4.4.8) は密 B 行列および演算結果のストア先である C 行列の制約条件を表している. 制約条件の内容は基本的に疎行列 A の制約条件と同じだが, 密行列 B および密行列 C

は CHIP 分割して実行する可能性があるので, 制約サイズを NCHIP で割った. また,BGRP と CGRP の条件は基本的に連動する必要がある. 密行列 B と密行列 C を別々に確保することは可能であるが, 仮に密行列 B の方が GRP を多く確保できたとしても, ストア先の密行列 C が最終段の LMM にない場合に, 対応する密行列 C の書き換えおよび前回の LMM の内容を主記憶に書き込みする必要があり, 実行時間はむしろ増大することが予想される. 図 4.9 からも CGRP と BGRP が連動していることはみてとれる. ゆえに式 (4.4.8) の制約条件を課した. 式 (4.4.9) の制約条件は密行列 B を LMM に確保するサイズの幅が IMAX の実行幅 W を超えないようにするためである. 先行研究で述べたように IMAX は物理的に横幅を持っているわけではなく, マルチスレディングにより仮想的に実現している. また, 掛ける 2 をされているのは,SIMD 実行で一度に 2set の計算を行うためである. 式 (4.4.10) は密行列 B を LMM に転送する際に BGRP×NCHIP が密行列 B の行サイズで割り切れるにして A のように Pad するコストを削減している.

$$pad = \begin{cases} 0 & (Acolumn \bmod H) == 0 \\ -Acolumn \bmod H + H & (Acolumn \bmod H) \neq 0 \end{cases} \tag{4.4.1}$$

$$Arow \times AGRP \times 2 \times 4 <= \frac{LMMSIZE}{2} \tag{4.4.2}$$

$$Arow \times AGRP \times 2 \times H <= Arow \times (Acolumn + pad) \times 2 \tag{4.4.3}$$

$$Brow \times BGRP \times 4 <= \frac{LMMSIZE}{2} \tag{4.4.4}$$

$$Brow \times BGRP <= \frac{Bcolumn \times Brow}{NCHIP} \tag{4.4.5}$$

$$Ccolumn \times CGRP \times 4 <= LMMSIZE \tag{4.4.6}$$

$$Ccolumn \times CGRP <= \frac{Crow \times Ccolumn}{NCHIP} \tag{4.4.7}$$

$$BGRP = CGRP \tag{4.4.8}$$

$$BGRP \neq W \times 2 \tag{4.4.9}$$

$$Bcolumn \bmod (BGRP \times NCHIP) \neq 0 \tag{4.4.10}$$

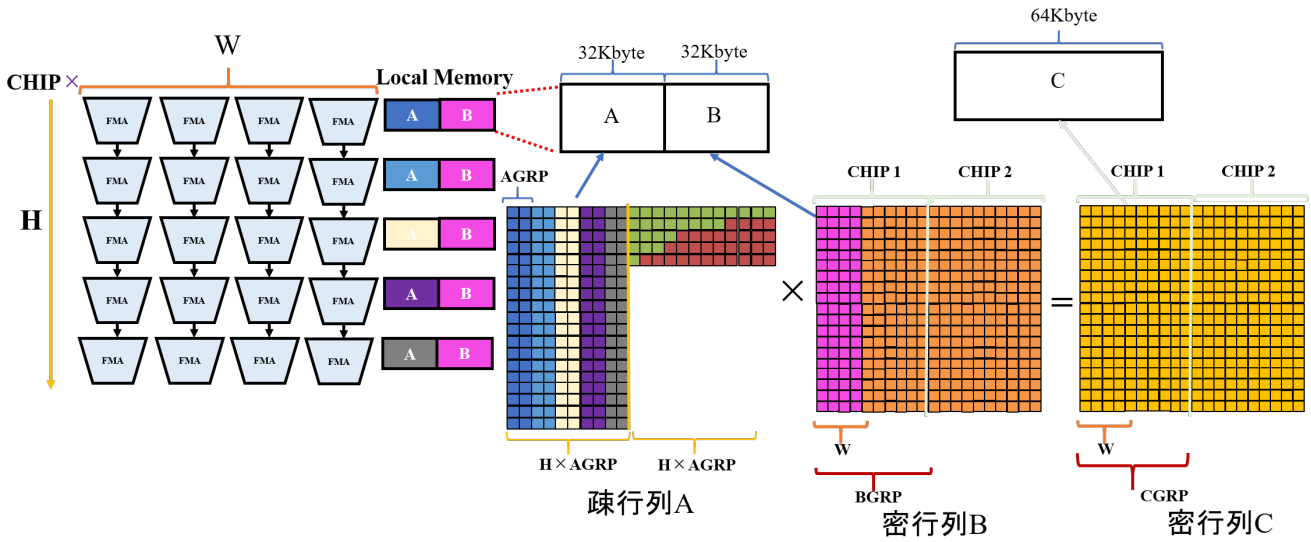

図 4.9　IMAX のパラメーター割り当て

第 5 章　評価と考察

5.1　評価環境

本説では，IMAX の FPGA プロトタイプシステムについて説明する．IMAX を RTL 記述し動作検証を行うため，FPGA SoC である Xilinx Zynq UltraScale+ ZU9EG を搭載する ZCU102 と，大規模 FPGA である Virtex Ultra Scale XCVU440 を搭載する S2C Single VU440 Prodigy Logic Module (VU440) を用いて ARMv8 + IMAX のプロトタイプを実装した．FPGA SoC と大規模 FPGA を用いた ARMv8+IMAX プロトタイプシステムの実装について説明する．プロトタイプのブロック図と実機の写真を図5.1 および，図5.2 に示した．図5.1 では，矢印の始点が Master，終点が Slave としている．ここで，Systolic Array は IMAX を指している．FPGA SoC である ZCU102 は，ARM を中心とする Processing System (PS) と，FPGA である Progmrammable Logic (PL) から成る．プロトタイプシステムは ZCU102 の ARM をホストとして，VU440 に実装した IMAX にアクセスする形をとっている．これは，ZCU102 の PL の規模では 64 段構成の IMAX を実装することが出来ず，大規模な FPGA が必要であったためである．ZCU102 - VU440 間は Xilinx の高速差動シリアル IO トランシーバーである GTH を 3 レーン用いて接続している．プロトタイプシステムの開発に使用した開発環境と，実装に用いた IP コアのうち，主要なものの一覧を表 5.1 に示した．これらの IP コアは，Xilinx 社から提供されており，Xilinx 社の FPGA を用いた開発において自由に利用することができる．

表 5.1　IP コア一覧

IP コア名	バージョン
Zynq UltraScale+ MPSoC	3.0
AXI Interconnect	2.1
AXI Chip2Chip Bridge	4.3
Aurora 64B66B	11.2

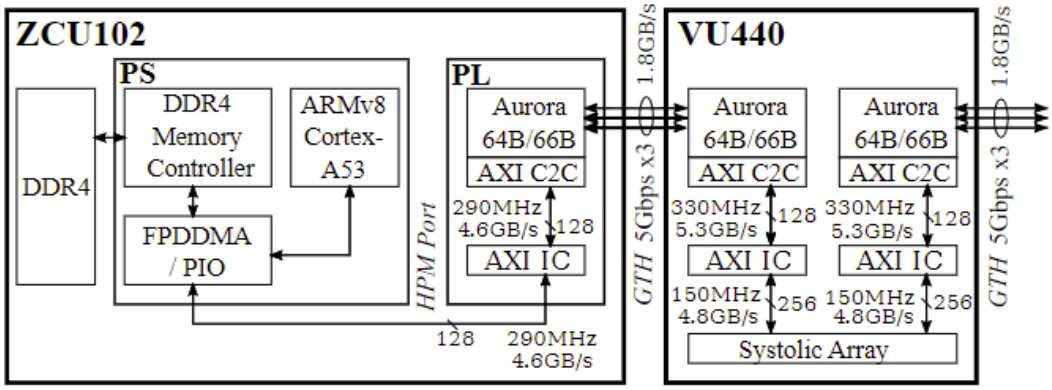

図 5.1　ARMv8 + IMAX FPGA プロトタイプシステムのブロック図

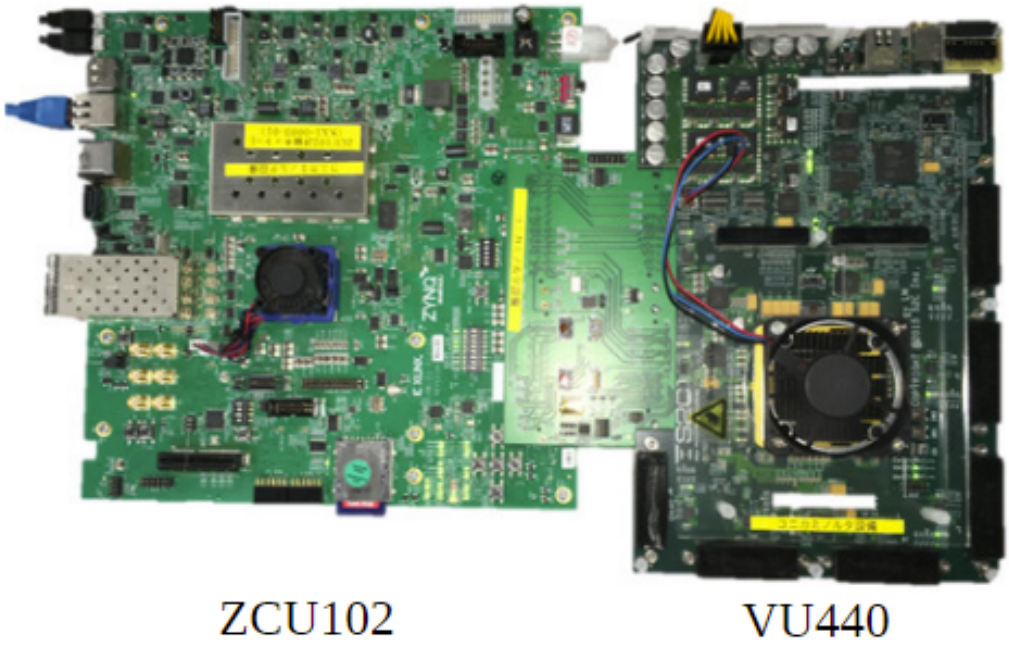

図 5.2　ARMv8 + IMAX FPGA プロトタイプシステム

5.1.1 ZCU102 へのホスト機能実装

まず，ZCU102 側の FPGA デザインについて説明する．PS には，FPDDMA という DMA エンジンが搭載されており，これを利用することで，広帯域幅な主記憶アクセスを実現する．ZCU102 の PL は，AXI Interconnect (AXI IC) を介して，チップ間通信を行うための IP コアである AXI Chip2Chip (AXI C2C) に接続し，64B/66B エンコーディングで GTH を使用できる Aurora 64B/66B を用いて外部と高速 IO を行うことができるデザインとなっている．C2C は ZCU102 側が Master である．AXI IC は PS 側，C2C 側共に 128bit で接続，330MHz で動作しており，理論帯域幅は 4.8GB/s となっている．ボード間の接続は，C2C の制約から，1 レーン 5Gbps で動作する GTH を 3 レーン使用しており，64B/66B エンコーディングなので，理論帯域幅は 5/8[GB/s]×3×64/66=1.818...[GB/s] となり，このボード間通信が本プロトタイプにおけるボトルネックである．

5.1.2 VU440 への IMAX 実装

次に，VU440 側の FPGA デザインについて説明する.VU440 側も ZCU102 側と同様に，Aurora 64B/66B を用いて GTH を 3 レーン使用し，C2C から AXI IC を介して IMAX に接続するデザインとなっている．理論帯域幅は C2C - AXI IC 間が 128bit 接続 330MHz 動作なので 5.3GB/s，AXI IC - IMAX 間が 256bit 接続 150MHz 動作なので 4.8GB/s である．また，マルチチップ接続に対応づけられる VU440 間に関しても ZCU102 側と同様に AXI IC と C2C，そして Aurora64B/66B を用いた GTH の 3 レーンを介して接続される．ここで，VU440 間も理論帯域幅は同様であるが，Master と Slave は接続の度に交互に入れ替わる．

5.1.3 使用するソフトウェアおよびアーキテクチャの詳細

本研究で使うソフトウェアおよびアーキテクチャについて説明する．表 5.2 に IMAX の環境について示した．OS には xilinx が配布する Debian 系の linux を使用した．gcc オプションは DMA を使用する都合上 O3 ではなく O2 を使用した．28nm で IMAX を製造した場合の Die Size を [13] に基づいて算出した.[13] では

表 5.2　IMAX の条件

OS	Debian 9.1
カーネル	4.9.0-xilinx-v2017.2
コンパイラ	gcc version 6.3.0
CPU	ARMv8 Cortex-A53 1.2GHz
gcc オプション	-O2 -mstrict-align
メインメモリ	DDR4-2133 single channel
アクセラレータ動作周波数	150MHz
アクセラレータ段数	64 stages
アクセラレータのロールメモリ容量	64k KB/ unit
Die size(mm^2)(28nm)	13.3

表 5.3　GPU の条件

GPU series	NVIDIA Pascal 256 CUDA Cores(Jetson TX2)
Architecture	NVIDIA Ampere 10496 CUDA Cores
Die size(mm^2)	43.6
nvcc option	-lcublas -lcusparse -$arch = sm\_62$
CUDA Version	10.2.89
GPU series	GeForce RTX 3090
Architecture	NVIDIA Ampere 10496 CUDA Cores
Die size(mm^2)	628
nvcc option	-lcublas -lcusparse -$arch = sm\_80$
CUDA Version	11.4
GPU series	GeForce RTX 3090

32kbye の LMM を使用していたが, 本研究では 64kbyte の LMM を使用するのでそれに伴う補正を行った. 表 5.3 に比較評価の際に使用する GPU の条件を示した. Jetson と Geforce とでは最適なオプションが異なるため,-$arch =$ でそれぞれ指定した.

5.2 疎行列積の評価

本節では,IMAX を用いた疎行列積の評価および考察を行う. ただし特に断りがない限り使用する IMAX の CHIP 数を 1 とする. 行列一辺のサイズが 1024,512,256,128,64 の正方行列どうしの行列積を評価に用いる. 例えば,1024×1024 は一辺 1024 の正方行列どうしの行列積を表している. Sparsity は行列要素のうち何 % が 0 かを表している. たとえば Sparsity=0.9 は 90% の要素が 0 であることを意味している. Speedup Ratio は密行列積での実行時間を疎行列積での実行時間で割ったものを表している. たとえば Speedup Ratio=2 は二倍の高速化を意味している. ここで図 5.3 に行列サイズごとの疎行列積の高速化率を示した. 行列サイズ 1024 から 256 の間で二倍以上の高速化を達成した. 行列サイズが大きいほど高速化率が上昇しており, 大規模なアプリケーションにも適応できることを示している. また Sparsity が 0.95 以降の高速化率上昇は疎行列を評価する上で重要な指標であり,1024×1024 の行列積では約 4.6 倍の高速化を達成している. 128 以下の行列サイズでは高速化を達成できなかった. これは演算量の削減よりもアドレス演算のために index を余分に LMM へ送るコストの方が高いことを意味している. 次に, 図 5.4 に総実行時間を 1 とした行列サイズ 1024,512,256 における密行列積と疎行列積の実行時間比率を示した. ただし疎行列積での Sparsity は 0.9 とする. 各内訳の動作に関する詳細は先行研究の章で説明した通りである. どの行列サイズにおいても密行列積では EXEC が最も実行時間を要していたのに対し, 疎行列積ではどの行列サイズにおいても EXEC の割合が密行列積と比べて半分以下に減少しており, Main Mermoy への書き戻し動作である DRAIN が相対的に増加している. これは Load の絶対量が Sparsity の増加で減り, 計算量が減るのに対して,Store する量は変わらないためである. つぎに図 5.5 に GPU との比較を示した. ただし GPU は初回実行時のレイテンシーが大きく, 金融やリアルタイム性が求められるアプリケーションに向いていないとされているが, 今回は 200 回連続実行してその平均を取ることでその遅延による影響を排除した. また密行列積と疎行列積のパフォーマンスを最大限に引き出すためにそれぞれ Nvidia が配布している専用ライブラリである Cublas と Csuparse を使用した. また IMAX の実行時間には Memory の転送時間やコンフィグ時間を含んでいるため,GPU 側のベンチマークもメモリーの転送動作を行う CudaMemcopy 関数を含んで計測した. Sparsiy が

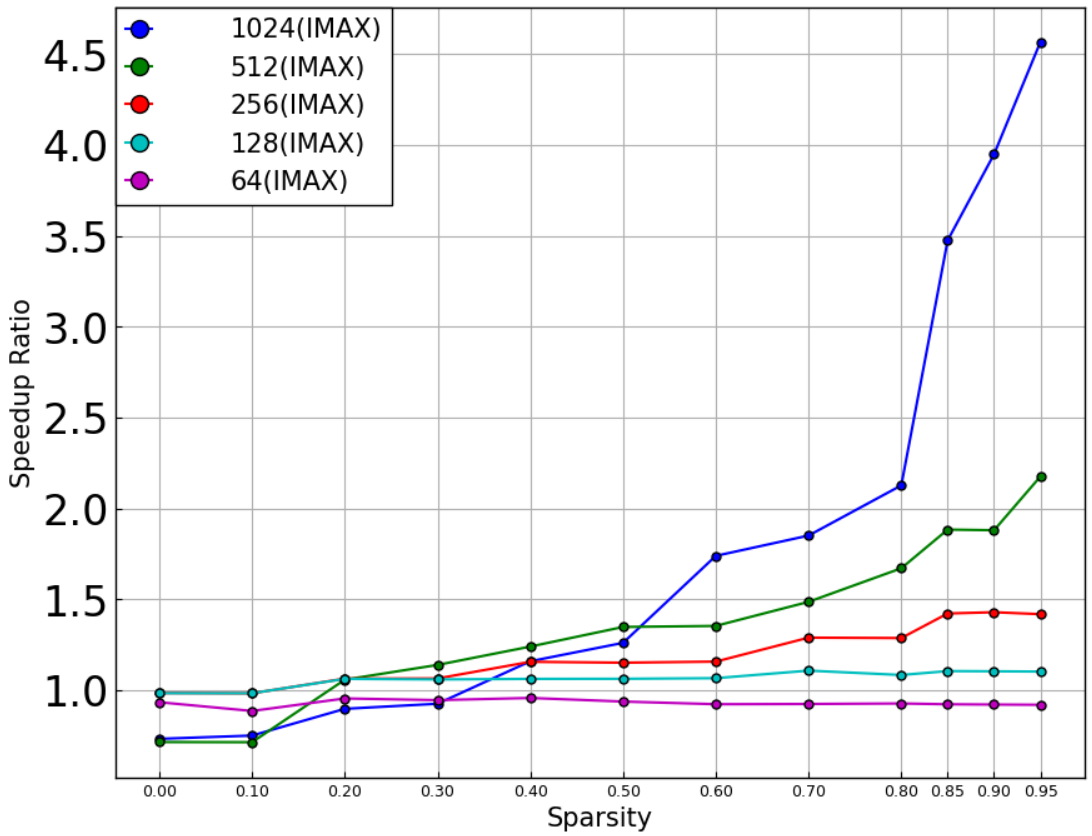

図 5.3　Sparsity ごとの疎行列積の高速化率

0.3 以下の領域では Speedup Rate がほぼ横並びであるのに対して,0.3 以降では IMAX の Speedup Rate が GPU の Speedup Rate を上回っているのを見てとれる．これは IMAX が GPU よりも疎行列乗算で優れていることを示している．ただし Geforce3090 に関してはアーキテクチャの規模に比べて対処の行列サイズが小さいため, 十分に性能を発揮できなかった可能性がある．

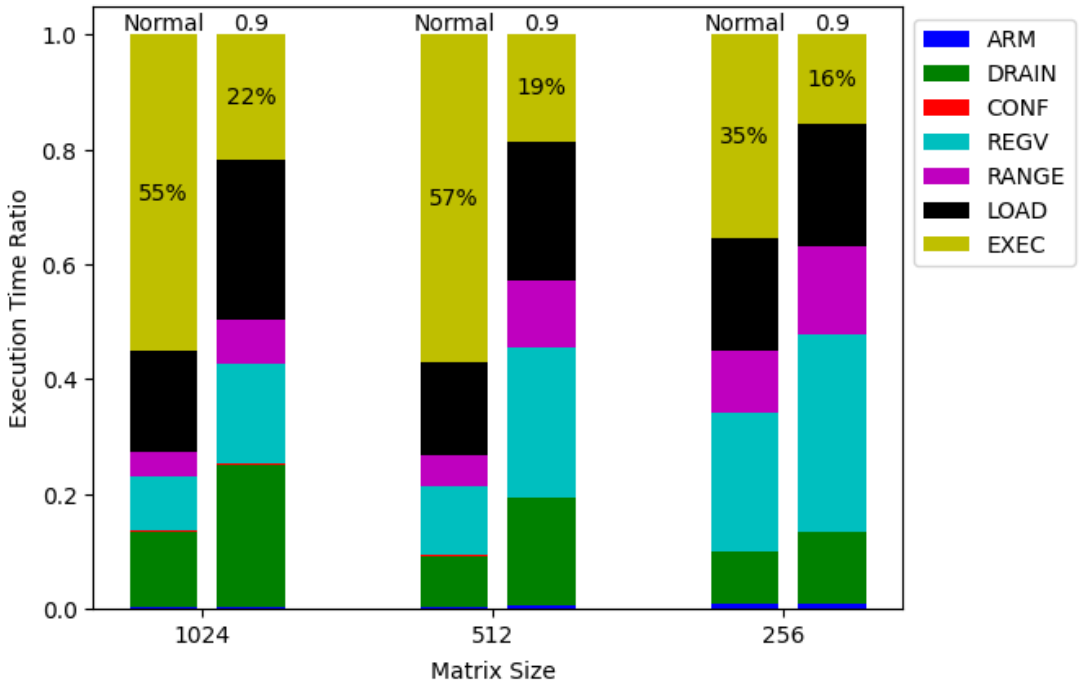

図 5.4　密行列積と疎行列積の実行時間比率

5.3　Local Memory 最適化に関する評価および考察

本節は前節で定義した評価環境を引き続き使用するものとする．本節では前章の章 4.3 で提案した疎行列をグループ化分割して転送する手法を適用し, なおかつ章 4.4 で定義した制約式に基づいて LMM の最適化を行った結果を評価する．ここで,LMM 最適前の LMM 使用率と最適化後の LMM 使用率を比較した結果を図 5.6 に示す．LMM の最適化前は LMM 使用率が最大 60% 程度なのに対して,LMM 最適化後は行列サイズ 1024 および 512 において 100% を達成している．これは疎行列をグループ化分割したことにより, 疎行列側の LMM を最大まで活用できたためである．さらにこの結果は人手によるチューニングではなく, 章 4.4 で定義した制約式の中を探索した結果によるものである．制約式の範囲内は探索範囲が狭く, 全探索が十分行えるため常に最適な結果を得ることができる．行列サイズが小さくなるにつれて LMM 使用率が落ち, 最適化前に近づいているのは行列の実行単位が小さくなることでグループ化するとはみ出てしまうことや一度ですべての要素が LMM

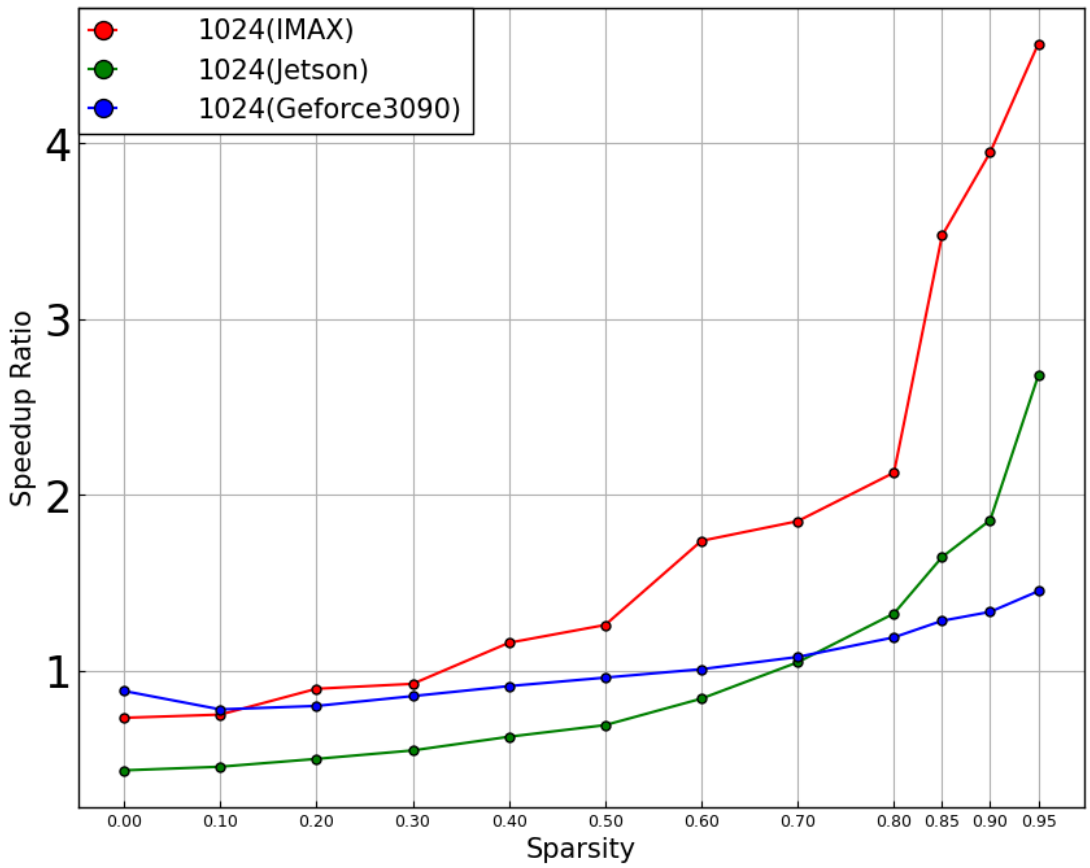

図 5.5　GPU との比較

に収まってしまうためである.

次に各行列サイズにおける LMM 最適化前と LMM 最適化後の比較図を図 5.7 に示した. すべての行列サイズで LMM 最適化による演算時間の削減を達成していることが見てとれる. 疎行列グループ化が行列サイズを問わずに有用であることが示された. Sparsity が上がるにつれて最適化前と最適化後の差が縮んでいるのは AGRP=1 ですべての疎行列が LMM に格納できるので, 最適化後前と同じスケジューリングになるためである. 最後に密行列積, LMM 最適化後の密行列積, LMM 最適化後の疎行列積の実行時間を比較した結果を図 5.8 に示した. ただし疎行列の Sparsity=0.95 とする. 図 5.8 より行列サイズが 1024 の時, 最適化前の密行列と最適化後の密行列を比較して, 計算時間を 67.5% 削減することを確認した.

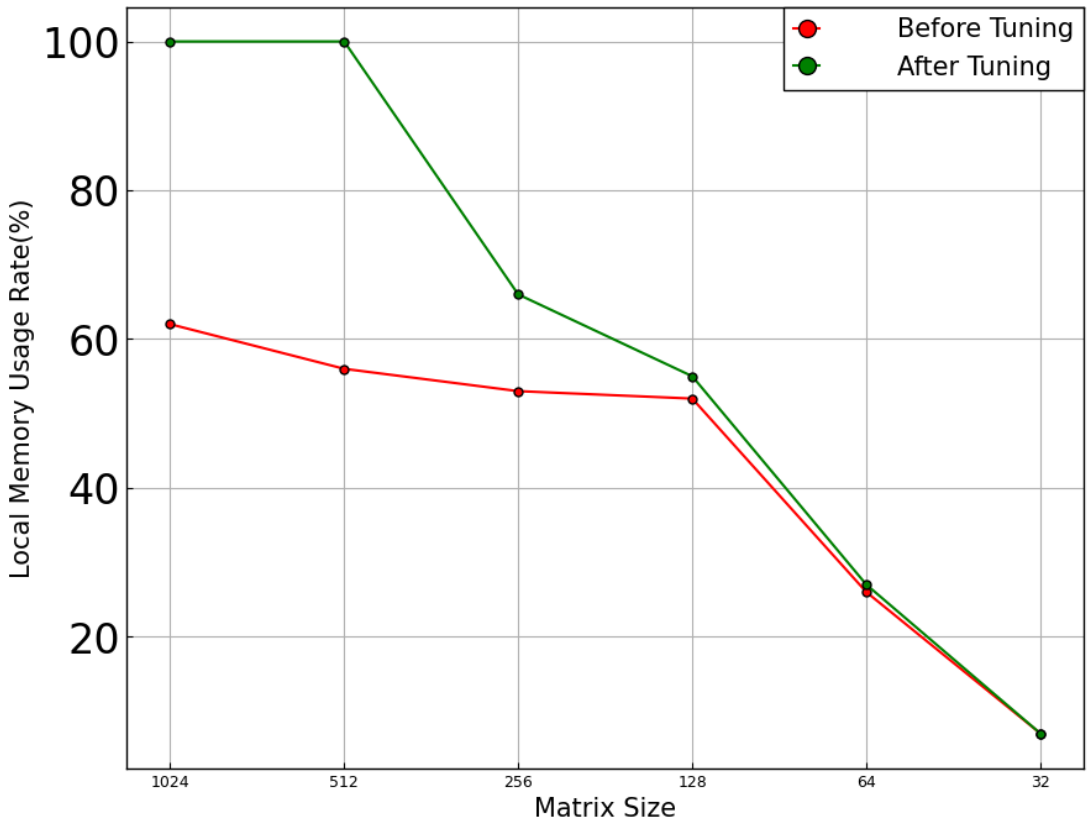

図 5.6　Local Memory の使用率

図 5.8 より行列サイズが 1024 の時, 最適化前の密行列と最適化後の疎行列を比較して, 計算時間を 92.9% 削減することを確認した.

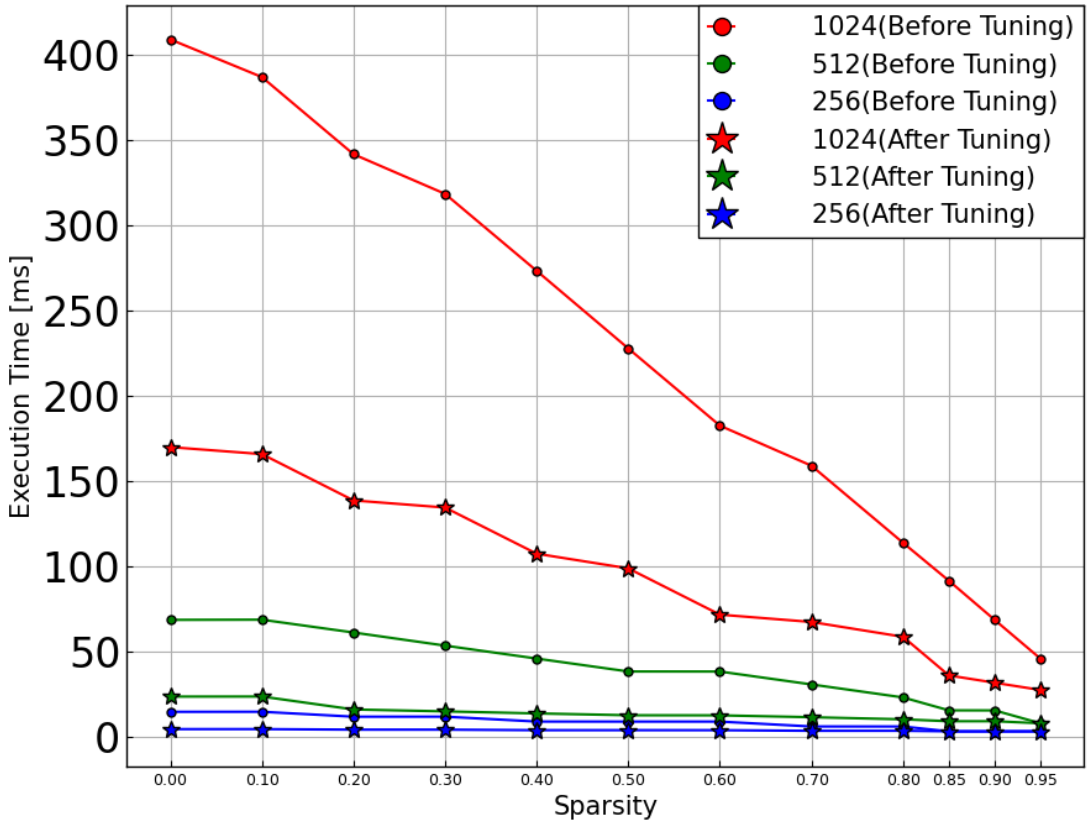

図 5.7　各行列サイズにおける Local Memory 最適化前と最適化後の比較

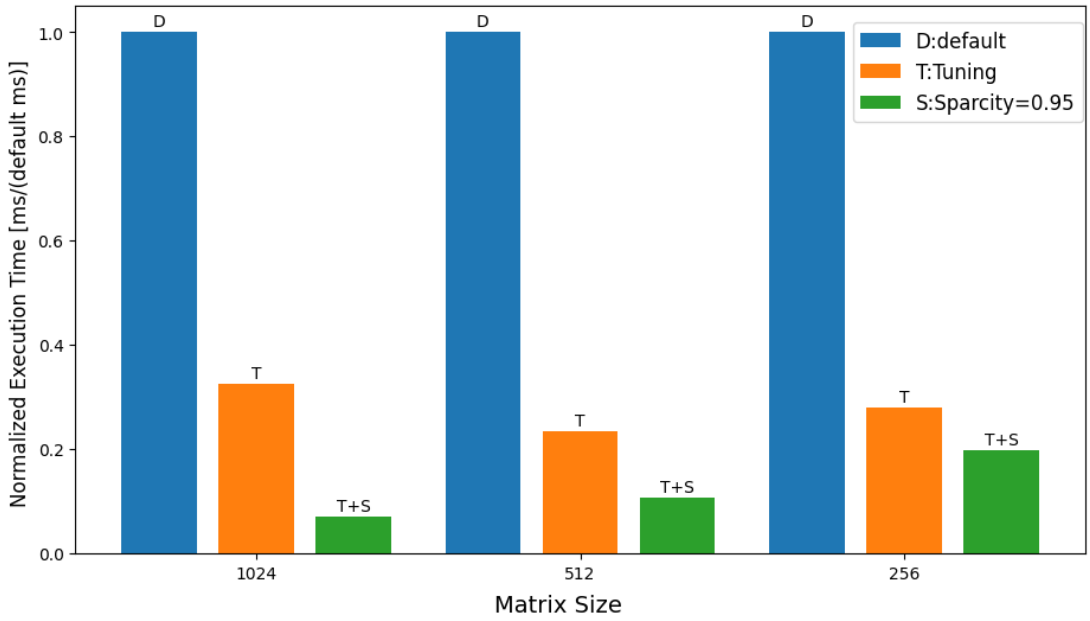

図 5.8　密行列と密行列最適化と疎行列最適化後の実行時間比較

5.4 偏りのあるデータに対する評価および考察

本節では偏りのあるデータに対する IMAX での評価および考察を行う．簡単のため, 本研究で偏りのあるデータとは行ごとの非ゼロの個数が異なるデータのこととする．本研究で使用した疎行列は基本的に C 言語の共有ライブラリに含まれている疑似乱数生成器を基に作成しており, データの偏りは少ない. しかし現実世界では偏りのあるデータが数多く存在しており, それらのデータに対処する必要がある. しかしスタンダードな CSR フォーマットでは偏りのあるデータの影響で疎行列をグループに分解できず, 性能低下が予想される．しかし本研究では JDS フォーマットを基に Margin などの要素を加えることで偏りのあるデータに対して対処をしている．ここで, 図 5.9 にデータの偏りを増やしていった際に JDS と CSR とでどのような差異が生まれるかを示した．ただし Sparsity=0.5 のままとし,Biased Ratio はデータの偏り率を表している．図 5.9 より,JDS フォーマットに Margin 要素を加えた手法の方が実行時間増加を抑えていることが見てとれる．このことから, 提案手法が CSR よりもデータの偏りに対してロバストであるといえる.

5.5 面積効率の評価および考察

本章の最後に疎行列と LMM 最適化によって面積効率が改善するかを GPU と比較しながら検証する．ここで図 5.10, 図 5.11 にそれぞれ密行列積の実行時間および面積効率の比較, 疎行列積の実行時間および面積効率の比較を示した．ただし面積効率は IMAX を 1 としたときの比較とする．また, 疎行列積を評価する際は Sparsity=0.95 とする．IMAX および GPU の面積効率を算出する際は表 5.1.3 で記載した Die Size を使用した．また 28nm での実行時間を算出する際は [13] を基に推定した．図 5.10.b では密行列でもすでに GPU に対して面積効率を上回っていることを示しているが, 図 5.11.b ではさらに上回っていることが見てとれる．これは, 疎行列積の計算において GPU より IMAX が優れていることを示している.

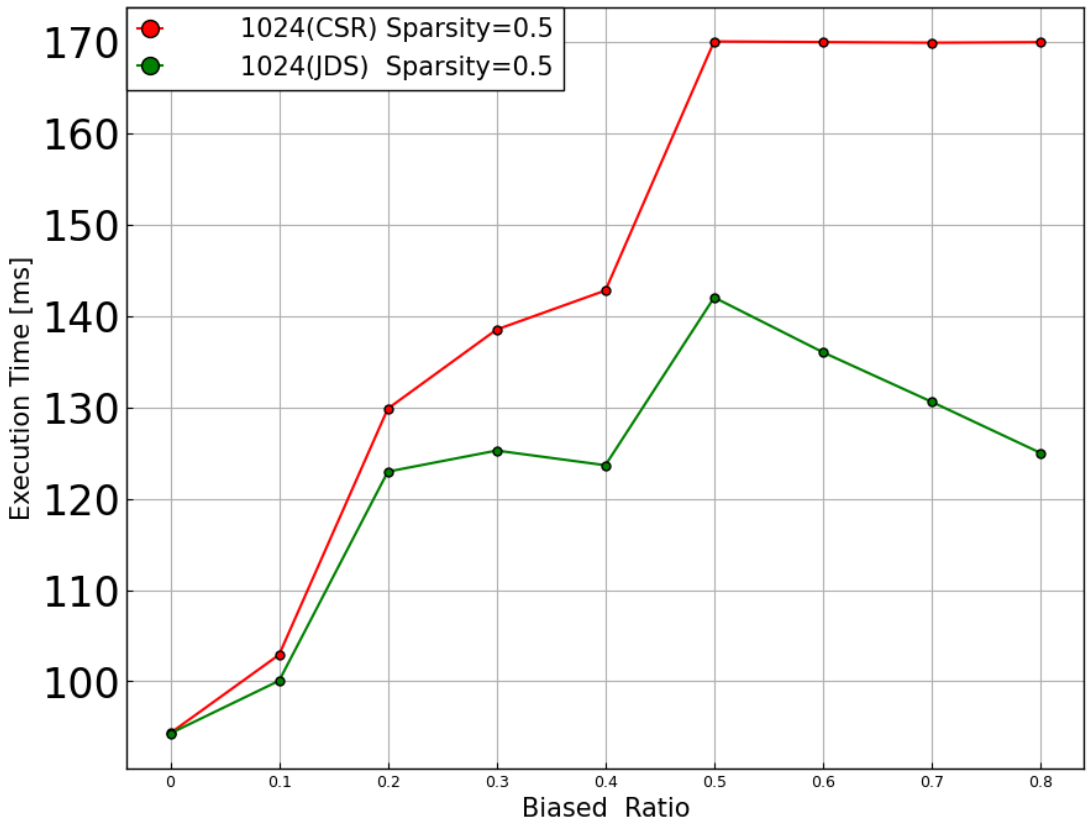

図 5.9　CSR と JDS の比較

5.6　疎行列ベクトル積の評価および考察

最後に疎行列ベクトル積の評価も行った．疎行列ベクトル積とは乗算行列の行サイズが 1 の行列積のことである．現実世界で得られるデータセットを集めた Florida Sparse Matrix Collection[23] から 10 個のデータセットを取得し, 疎行ベクトル積のベンチマークデータとして使用した．疎行列ベクトル積は HPC 分野のベンチマークとして使われることが多く, 多くのアプリケーションのボトルネック部分となっている. 本研究で提案した手法が疎行列積のみならず, 疎行列ベクトル積でも有用なことを示す．ここで,CSR でデータセットを計算した場合の結果を図 5.12 に示した．reorientation_7 以外のデータセットで, 計算時間の削減に成功した．しかし,reorientation_7 では Sparsity が高いのにも関わらず, 密行列より

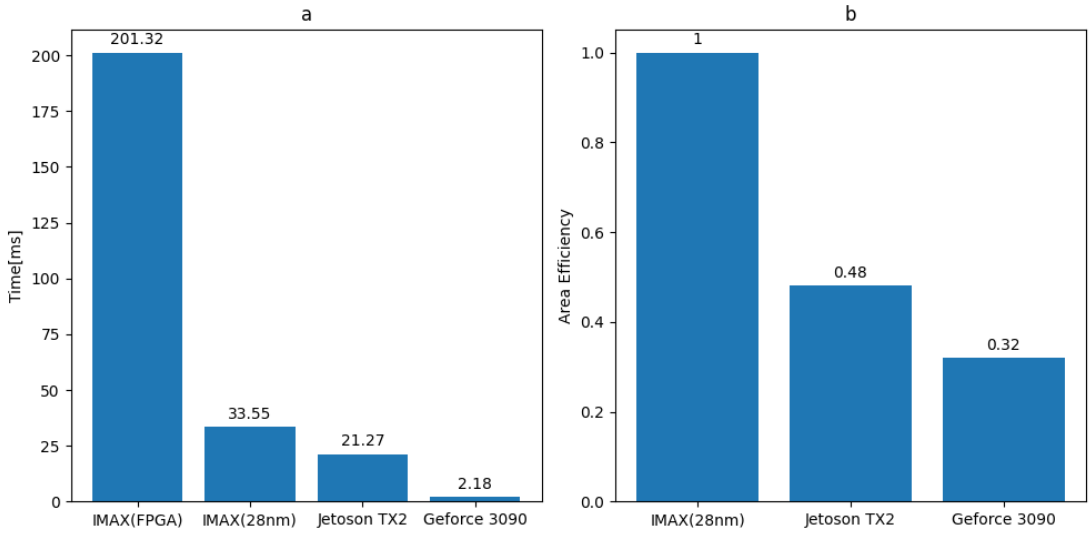

図 5.10　密行列積の実行時間および面積効率の比較

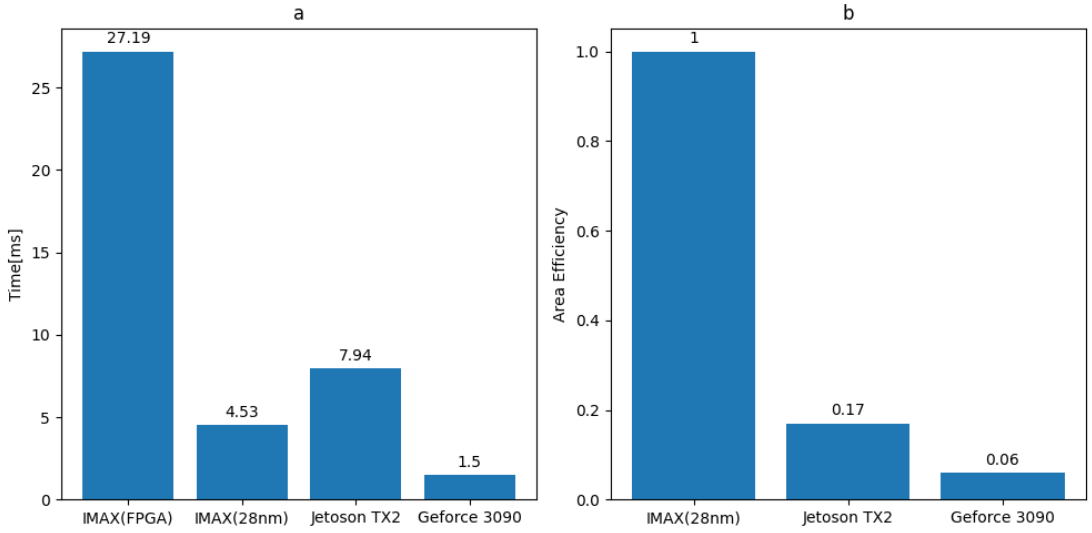

図 5.11　疎行列積の実行時間および面積効率の比較

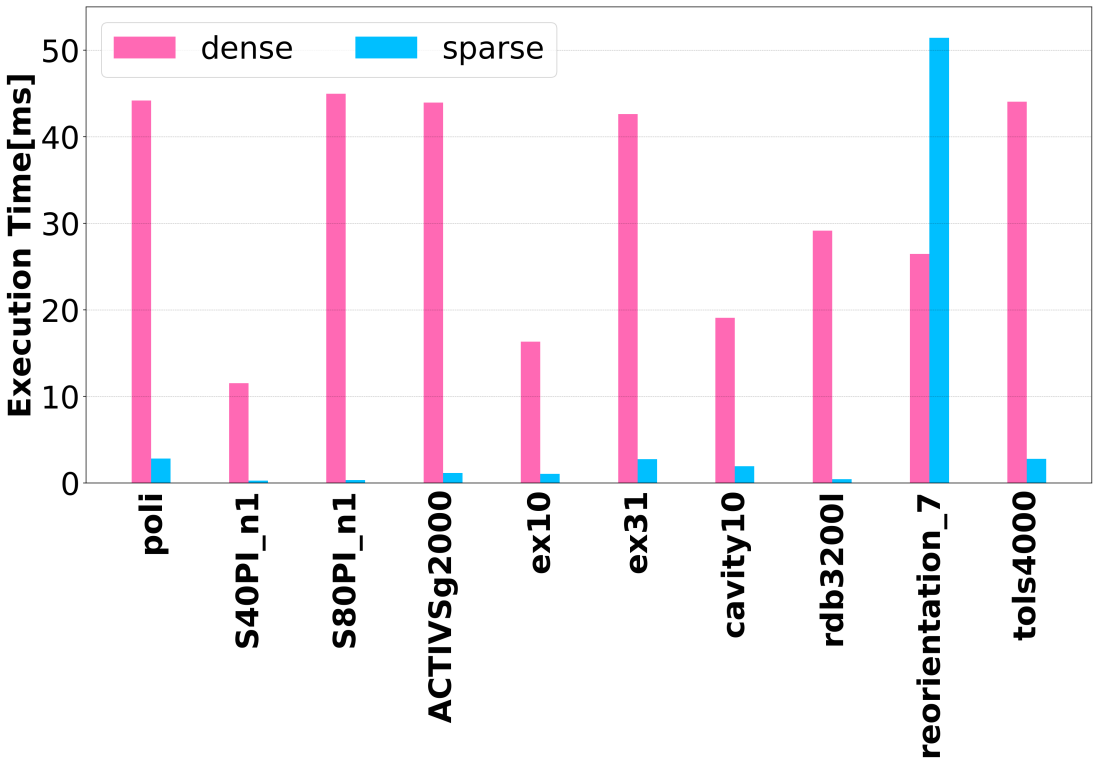

図 5.12　CSR による疎行列ベクトル積の実行結果

計算時間が増加した．ここで,reorientation_7 を可視化した図を図 5.13 に示した．図 5.13 より, 縦と横に線が入っており, ブロッキングするのを妨げている．これに対処するために疎行列積の時と同様に JDS を適応し, 並べ替えを行った．並べ替え後に計算した結果を図 5.14 に示した．図 5.14 の結果より,reorientation_7 での性能低下を軽減していることを見てとれる. 最終的に, 平均で 94.3% の計算量削減に成功した.

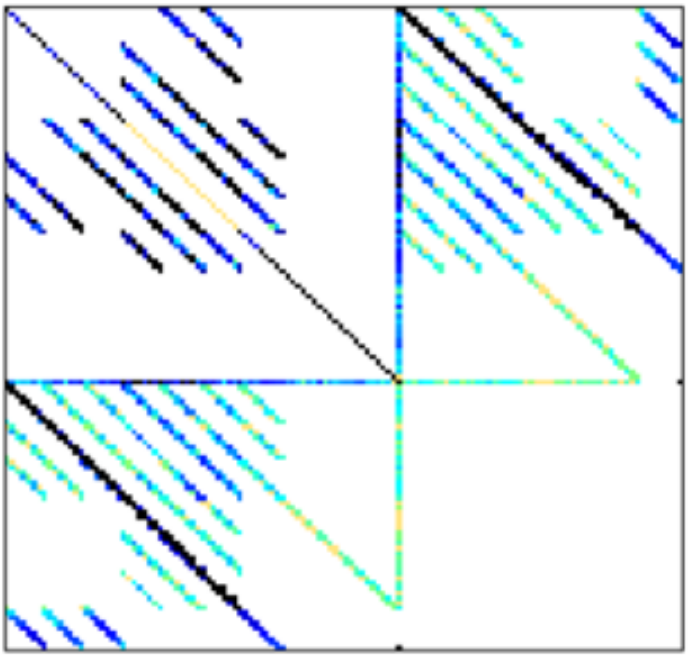

図 5.13　reorientation7 の可視化内容

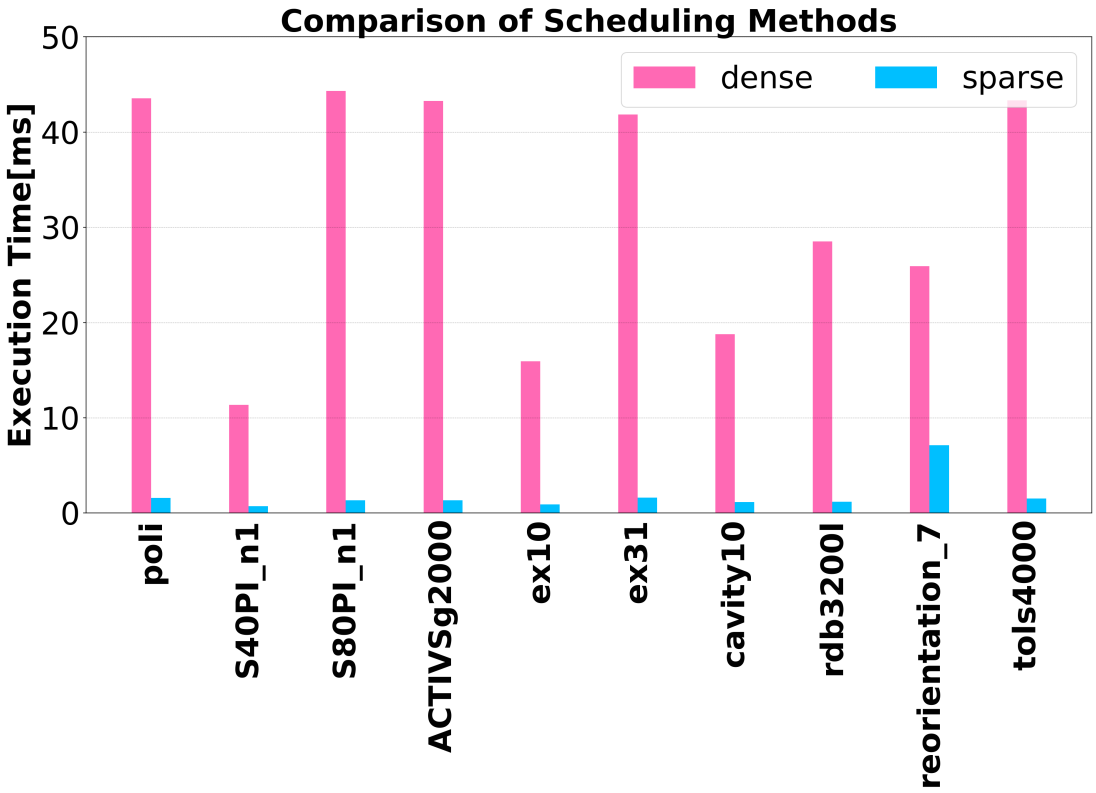

図 5.14　JDS による疎行列ベクトル積の実行結果

第 6 章　おわりに

本研究では IMAX を用いた疎行列積の高速化および LMM の最適化を提案した. JDS に Margin 要素を加えた独自フォーマットを二通り提案し, フォーマットを最大限に活かすようなスケジューリングを検討した. さらに値と index を一つにまとめる IVJ 法が IVS 法よりも効率的なことを明らかにした．またデータの偏りに対してもロバストなことを示し, 現実データへも適応できる可能性を示した．疎行列積を演算する際に疎行列側をグループ化して LMM へ転送することにより LMM の使用率向上を達成した. さらに IMAX のハードウェアパラメータの制約を定式化し, それの範囲内で探索を行うようなチューニングをすることで人手の介在なして最適化することまでも可能にした．GPU と比較して面積効率で優位性があることを示し, 疎行列積ではその優位性はさらに増大することを示した．疎行列ベクトル積を実データで検証し, 並べ替えでコーナーケースにも対応できることを示した. 最終的に密行列積 1024×1024 を従来手法で行った実行時間と比較して,Sparsity が 95% の際に計算時間を 92.9% 削減することを示した．また,10 個の実データセットを用いて疎行列ベクトル積をベンチマークを行い, 平均で計算時間を 94.3% 削減することを示した.

謝辞

研究活動はもちろんのこと，研究室での交流に至るまで常日頃より的確なご助言やご指導，ご支援を頂いた本学の中島康彦教授及び Renyuan ZHANG 準教授に心より深く感謝の意を表します．本論文をご精読いただき，また発表に対して貴重なご意見を頂いた本学の林 優一教授に深く感謝したします．ミーティングなどでお世話をしていただきました中田尚元准教授に深く感謝いたします．研究活動にとどまらず, 仮想通貨の運用についてもご指導してくださった野村武司氏，2 年間の研究生活や私生活で多くを支えてくださった同期の赤部知也氏, 稲益 秀成哉氏，澤田 篤志氏に深く感謝いたします．そして普段より研究室の運営維持のために多くの係を引き継いでくださった押尾怜穏氏，船井遼太朗氏ら研究室の後輩たちに深く感謝いたします．最後に支えてくれた家族に心より感謝申し上げます．

参考文献

[1] Nvidia, "NVIDIA A100 Tensor Core GPU," *White Paper*, pp. 20–21, 2020.

[2] J. Fowers, K. Ovtcharov, K. Strauss, E. S. Chung, and G. Stitt, "A high memory bandwidth FPGA accelerator for sparse matrix-vector multiplication," *Proceedings - 2014 IEEE 22nd International Symposium on Field-Programmable Custom Computing Machines, FCCM 2014*, pp. 36–43, 2014.

[3] S. Pal, J. Beaumont, D. H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H. S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator," *Proceedings - International Symposium on High-Performance Computer Architecture*, vol. 2018-February, pp. 724–736, 2018.

[4] H. T. Kung, "Why Systolic Architectures?," *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[5] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," *Proceedings - 2020 IEEE International Symposium on High Performance Computer Architecture, HPCA 2020*, pp. 58–70, 2020.

[6] N. Whitepaper, "Sparsity Enables 100x Performance Acceleration in Deep Learning Networks A Technology Demonstration," 2021.

[7] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, 2019.

[8] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," *Advances in Neural Information Processing Systems*, vol. 2020-December, no. NeurIPS, 2020.

[9] A. Mamri, M. Abouzahir, M. Ramzi, and R. Latif, "ORB-SLAM accelerated on heterogeneous parallel architectures," *E3S Web of Conferences*, vol. 229, pp. 1–7, 2021.

[10] Y. Zhu and D. Mutchler, "On constructing the elimination tree," *Discrete Applied Mathematics*, vol. 48, no. 1, pp. 93–98, 1994.

[11] Y. M. Tsai, T. Cojean, and H. Anzt, "Evaluating the Performance of NVIDIA's A100 Ampere GPU for Sparse Linear Algebra Computations," 2020.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. Richard Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *Proceedings - International Symposium on Computer Architecture*, vol. Part F128643, no. March 2018, pp. 1–12, 2017.

[13] J. Iwamoto, Y. Kikutani, R. Zhang, and Y. Nakashima, "Daisy-chained systolic array and reconfigurable memory space for narrow memory bandwidth," *IEICE Transactions on Information and Systems*, vol. E103D, no. 3, pp. 578–589, 2020.

[14] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx adaptive compute acceleration platform: Versal™ architecture," *FPGA 2019 - Proceedings of the 2019 ACM/SIGDA International Symposium on Field-*

Programmable Gate Arrays, pp. 84–93, 2019.

[15] Y. H. Chen, T. J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[16] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks," *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 553–564, 2017.

[17] B. Arcelin, "Comparison of Graphcore IPUs and Nvidia GPUsfor cosmology applications," pp. 1–11, 2021.

[18] I. T. Ag, "CUSTOM IMPLEMENTATION OF THE COARSE-GRAINED RECONFIGURABLE ADRES ARCHITECTURE FOR MULTIMEDIA PURPOSES Advanced Systems and Circuits Kapeldreef 75 , Leuven , Belgium," pp. 106–111, 2005.

[19] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer, "PipeRench: A coprocessor for streaming multimedia acceleration," *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*, no. May, pp. 28–39, 1999.

[20] N. Ozaki, "C Ool M Ega -a Rrays : U Ltralow - P Ower R Econfigurable," pp. 6–18, 2011.

[21] N. Devisetti, T. Iwakami, K. Yoshimura, T. Nakada, J. Yao, and Y. Nakashima, "LAPP: A low power array accelerator with binary compatibility," *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 854–862, 2011.

[22] C. Nicol, "A Dataflow Processing Chip for Training Deep Neural Networks Chief Technology Officer Wave Computing," *Hot Chips*, 2017.

[23] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, 2011.

発表リスト

[1] S. Takuya, R. Zhang and Y. Nakashima, "Training Low-Latency Spiking Neural Network through Knowledge Distillation," 2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), 2021, pp. 1-3, doi: 10.1109/COOLCHIPS52128.2021.9410323.