

DRAM計算を活用した行列ベクトル積の エンドツーエンドFPGAアクセラレータ

勝見 舜[†] 久保 龍哉^{†,††} 篤田 大知[†] 三好 健文^{†††} 高前田伸也^{†,††}

[†] 東京大学

^{††} 理化学研究所

^{†††} わさらば合同会社

あらまし データ中心アプリケーションの普及に伴い、DRAMの高い記憶密度を維持しつつメモリ内部で並列性の高い演算を行う Processing-Using-DRAM (PUD) が注目を集めている。しかし、従来の PUD では計算の柔軟性の乏しさに起因するデータ転送量の増大や、ビットシリアル演算による高いレイテンシが課題であった。本研究ではこれらの課題を解決するため、PUD とプログラマブルロジックを協調させるシステムを提案し、広範なデータ中心アプリケーションで要求される大規模かつ低ビットの行列ベクトル積の高速化を実現する。市販の無改変な DDR4 DRAM を用いた実機評価を行い、メモリ帯域幅の理論値を超える計算スループットを達成することを実証する。

キーワード Processing-Using-DRAM, 動的 PUD, DDR4 DRAM, 行列ベクトル積, FPGA, 高位合成

1. はじめに

データ駆動型アプリケーションの急激な規模拡大に伴い、システムにおいてデータ密度を確保することは必要不可欠である。しかしながら、大量のデータを記憶できることは要件の一側面にすぎず、実際のシステムではそのデータを用いて高スループットに計算を実行できなければならない。そこで、Processing-Using-DRAM (PUD) は、現在主記憶として広く用いられている DRAM の高いデータ密度を維持しつつ、メモリセルアレイ内で直接計算を行うことでメモリ内部の膨大な潜在的並列性を活用できるアプローチとして注目されている [5], [10], [11]。メモリ近傍に従来型の演算器を導入する Processing-Near-Memory (PNM) アーキテクチャ [4], [8] とは対照的に、演算に特化した回路へのデータ転送は行われず、メモリセル上で自然に実現されるビットシリアル演算を行うのが特徴である。

しかし、従来の PUD アーキテクチャは PUD で計算を行うために、かえって DRAM とプロセッサ間のデータ転送量が肥大化してしまうという問題があった。これは、PUD の非柔軟な性質にも関わらず、DRAM をスタンドアロンなアクセラレータカードとして捉えていたことが根本的な原因である [5], [9], [11]。具体的には、PUD はビットシリアル計算であるため浮動小数点等の複雑な計算ではレイテンシの増加が避けられず、またメモリセルアレイのカラム間でのデータの行き来に大きな制約があることにより、計算やデータ転送が冗長になる問題がある。それにも関わらず、従来は PUD 内で処理が完結するという非現実的なワークロードが想定されていたため、冗長なデータ転送を開始時に行い、GPU のカーネルのように固定されたプログラムを PUD 向けの DRAM コマンド列として転送し、やはり冗長な計算結果をプロセッサに戻

すというフローが採用されており、実世界のワークロードを十分に高速化できない状況に陥っていた。

PUD は並列性は膨大だが計算の柔軟性が低い。この特性を活かすためには、プロセッサと PUD が協調して、システム全体としてアクセラレーションを行う枠組みが必要がある。この枠組みにおいては、プロセッサ側からのデータの転送や PUD 計算は動的に行われ、また動的に最適化される。本稿では、この枠組みを**動的 PUD**と呼ぶ。先行研究の MVDram は、データ駆動型アプリケーションにおいて頻繁に用いられる大規模・低ビットの行列ベクトル積演算 (MVM) を対象とし、プロセッサと PUD が協調することで初めてメモリバンド幅を超える可能性を持つ PUD アルゴリズムを提唱した [7]。具体的には、MVM の一部として入力ベクトルに依存した動的な行列圧縮計算を DRAM にオフロードし、残りはプロセッサが行う手法であった。しかしながら、実システムとしての実装および評価は行われていない。実際、動的 PUD によりボトルネックがデータ転送からビットシリアル演算実行のレイテンシに移ったことにより、高速な実行のためには (1) 演算の動的な実行および最適化と、(2) メモリ内のバンク並列性の動的な活用の 2 点が課題として残っている。(1) についてはどのような PUD 命令を発行するかをリアルタイムに決定しなければならず、その処理がボトルネックになってしまえば本末転倒である。また、ビットシリアル演算の高速化のために共通部分式除去 (CSE) といった最適化の適用可能性もオンラインで決定する必要がある。(2) については動的 PUD をメモリバンクごとに行うと、異なるバンクが異なる PUD 処理を行うという MIMD (Multiple Instruction, Multiple Data) のような状況が発生しうる。これを最大限に活用できる能力がプロセッサ側に求められる。

そこで本研究では、これらの課題を克服し、商用の DDR4 DRAM をそのまま用いしつつも、DRAM メモリバンド幅を凌駕

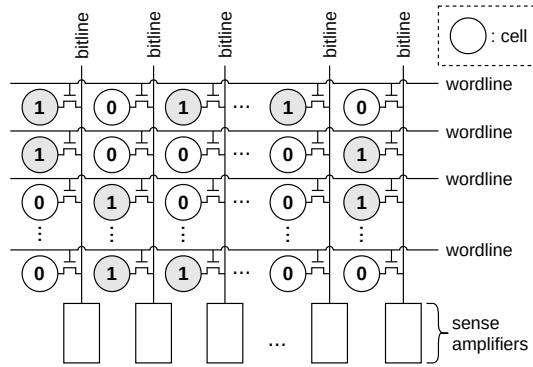


図1 DRAMの構造
Fig.1 DRAM structure.

する計算スループットを持つ実システムを提案する。動的なビットシリアル演算の実行およびリアルタイム最適化のため、プロセッサ側に配置される PUD 処理回路は高位合成を用いて FPGA 上に高性能に実装される。この回路においては、入力ベクトルのビット複雑性を活用した CSE のためのパターンマッチや Booth エンコーディング [1] を行う。メモリのバンク並列性を最大限活用するために、この PUD 処理回路は FPGA 上にバンク数と同じ数に展開され、並列に PUD 処理を実行できるアーキテクチャを採用する。また、PUD を可能とする FPGA 上のカスタムメモリコントローラ [13] も統合して活用する。これらによって DRAM 内部の膨大な並列性を最大限に引き出すことに成功した。

本研究の貢献は以下のとおりである。

- ・DRAM とプログラマブルロジックを協調させ、動的 PUD を行う実システムを初めて構築した。
- ・ビットシリアル演算の高速化のため、共通部分式除去や Booth エンコーディング等の最適化を模索した。
- ・動的 PUD により生じる MIMD バンク並列を最大限活用する処理回路を構築した。
- ・大規模かつ低ビットな行列ベクトル積においてメモリバンド幅を超える計算スループットを実証した。

2. 背景

2.1 DRAM

DRAM はビットを保持するメモリセルの 2 次元の配列からなる。1 つのトランジスタと 1 つのコンデンサーのみで 1 ビットの情報を保持でき (1T1C 構造) [2]、高速なアクセス性能を維持しつつ、データ密度が高い。配列はサブアレイという単位で構成され、図 1 のように主にワードライン、ビットライン、センスアンプからなる。DRAM にアクセスする際は DRAM コマンドをプロセッサ側から発行する。このコマンドは非常に粒度が細かく、DRAM 内の物理的な状態遷移に対応する。まずプリチャージ (PRE) を行うことで、センスアンプの電位をニュートラルに準備する。その後ワードラインを活性化 (ACT) することで、サブアレイの特定の行を導通させ、ビットラインを介してデータをセンスアンプに引き出す。その後読み取り・書き込みのコマンドを発行することでセンスアン

プに増幅されたデータの読み出しや書き換えが可能となる。

DRAM にはバンクという並列性と状態保持のための単位があり、これが複数のサブアレイを保持する。DDR4 では通常バンクは 16 個あり、これが並列に動作する。異なるバンクであればプリチャージや行の活性化を並列に行うことができる。しかし、コマンドの送信やデータの読み書きに使うバスはバンク間で共有される。

2.2 商用 DRAM での Processing-Using-DRAM (PUD)

DRAM コマンドはプロセッサ側から発行されるものの、DRAM 内部の物理的な安定化を待つためにタイミング制約が設けられ、仕様化されている [6]。しかし、この仕様を意図的に違反することで、商用の DRAM 内部で計算挙動が生じることが発見された [3], [12]。具体的には、ある行 A を活性化した後別の行 B を活性化する際には通常プリチャージ後、一定の時間間隔が必要であるが、この間隔を待たずに行 B を活性化することにより行 A のデータを行 B にコピーすることができる (RowCopy) ほか、より短い間隔で行 B を活性化すると行 A と行 B が同時に開放され、アナログ原理に基づいた多数決演算 (SiMRA: Simultaneous Multiple Row Activation) が可能であることが発見された。これらは通常 65536 あるカラムで同時に演算が行われるため、DRAM 内部の膨大な並列性をそのまま活用することができる。

ビット反転演算と多数決演算を合わせると完全性があり、任意の論理的関数を表現することができるが、ビット反転演算を同一サブアレイで行う方法は回路修正の無い DRAM では発見されていない。そのため、DRAM 各行に対しそのビット反転を常に保持しておくカップリング手法を取る手法が提案されている [3]。カップリングは多数決演算をビット反転行に対しても行うことで以下のように維持されるが、計算レイテンシが 2 倍になる。

$$\neg \text{MAJ}(A, B, C) = \text{MAJ}(\neg A, \neg B, \neg C)$$

PUD による計算は多数決演算に基づくビットシリアル演算となり、レイテンシが課題となることが多いほか、カラム間でのデータのやり取りが PUD のみでは不可能なため、場合によってはデータ配置や計算が冗長になってしまう制約がある。

2.3 MVDRAM

MVDRAM [7] は商用・未改変の DDR4 DRAM において PUD を行い、メモリバンド幅を超えた計算スループットを達成するためのアルゴリズムとして提案された。MVDRAM では行列データ転送量削減を念頭に、入力ベクトルに依存した動的な行列の圧縮のみが DRAM 内で行われ、残りの演算はプロセッサ側で処理される。本研究では、この提案アルゴリズムにおいて特徴である「動的 PUD」と「水平方向に行列を保存するレイアウト」を継承しつつ、より効率的な DRAM 内行列圧縮手法を提案し (3.)、またシステムレベルで実装を行い、実機での性能の検証結果を述べる。

3. PUD による DRAM 内行列圧縮

3.1 アルゴリズム

以下の計算を高速化することを目標とする．なお，行列の配置を DRAM 上での配置と合わせるため，ベクトルは転置されている：

$$\mathbf{v}_{\text{out}}^T = \mathbf{v}_{\text{in}}^T M$$

データ中心のアプリケーションを想定し，行列 M のサイズが十分に大きい場合を考える．その際，行列は主記憶に保存され，その行列の読み出しの帯域がボトルネックになることが多い (Memory Wall)．そこで，本研究では，プロセッサ側に保存された入力ベクトルの値を実行時に参照し，その値に応じて，行列を PUD によって DRAM 内で変形・縮小を行い，行列ベクトル積の実行で必要なデータ転送量を削減する．行列の変形に応じて入力ベクトルも対応する形に変形し，元の行列ベクトル積と等価な結果を保つ．具体的には，以下のような行列ベクトル積が与えられた際，等号を保ったまま行列（とベクトル）サイズが小さくなるようにする．

$$(1\ 2\ 3) \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} = (a\ b) \begin{pmatrix} c & d & e & f \\ g & h & i & j \end{pmatrix}$$

必然的に，行列のカラムごとに計算が行われ，PUD にマップしやすい．例えば，一番左の列では $1 \cdot 1 + 2 \cdot 5 + 3 \cdot 9 = a \cdot c + b \cdot g$ が保たれるようにする．

ここで，全加算器と全減算器を用いて圧縮を行うことにする．これらは 3 入力 2 出力の，以下の計算を行うバイナリレベルの演算を行う．

$$A + B + C_{\text{in}} = 2 \cdot C_{\text{out}} + S, \quad A - B - C_{\text{in}} = -2 \cdot C_{\text{out}} + S$$

全加算器のみでは符号が異なる場合に適用できないため，全減算器も併せて用いている．これらはバイナリレベルの演算であるが，行列の要素の各ビット位置に並列に適用することで任意のビット幅の値に対しても適用可能である．これらの演算を適宜等倍等することで，以下のように行列を小さくしていく．この過程をリダクションと呼ぶことにする．

$$\begin{pmatrix} 2 \\ 2 \\ 5 \\ -3 \\ -3 \end{pmatrix}^T \cdot \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \\ m & n & o \end{pmatrix}$$

$$= \begin{pmatrix} 2 \\ 2 \\ 2+3 \\ -3 \\ -3 \end{pmatrix}^T \cdot \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \\ m & n & o \end{pmatrix}$$

準備

$$= \begin{pmatrix} 4 \\ 2 \\ 0+3 \\ -3 \\ -3 \end{pmatrix}^T \cdot \begin{pmatrix} C_{\text{out}0} & C_{\text{out}1} & C_{\text{out}2} \\ S_0 & S_1 & S_2 \\ g & h & i \\ j & k & l \\ m & n & o \end{pmatrix} \quad \begin{array}{l} \text{上から 3 行に} \\ \text{全加算器適用} \\ \text{(係数: 2)} \end{array}$$

$$= \begin{pmatrix} 4 \\ 2 \\ -6 \\ 3 \end{pmatrix}^T \cdot \begin{pmatrix} C_{\text{out}0} & C_{\text{out}1} & C_{\text{out}2} \\ S_0 & S_1 & S_2 \\ C'_{\text{out}0} & C'_{\text{out}1} & C'_{\text{out}2} \\ S'_0 & S'_1 & S'_2 \end{pmatrix} \quad \begin{array}{l} \text{下から 3 行に} \\ \text{全減算器適用} \\ \text{(係数: 3)} \end{array}$$

リダクションの仕方はプロセッサ側で決定される．つまり，プロセッサ側は**行列の値を見なくとも**，保持しているベクトルの値のみの参照で全加算器・全減算器の適用を決定でき，リダクションを進めることが可能である．

リダクションにおいては上の例における「準備」のステップを効率的に行わなければならない．このステップでは，どのようなパターンでリダクションを進めれば効率的に行数が減らせるかを見通す必要がある．先行研究の MVDRAM [7] ではベクトルの各値をビットごとに分解して確実に処理していたが，この方法では冗長なリダクションが発生していた．本手法ではヒューリスティクスに基づく**パターンマッチ**の手法でベクトルの複数ビットにまたがったリダクションを行い，必要なリダクション回数を減らすことで性能を上昇させる．具体的には，ベクトル中の値を POPCNT やビット 1 に関する MSB (Most Significant Bit) 位置等を元にソートしておき，昇順に値を 3 つ取り出して，その値のビット論理積を計算し，その値を係数としてリダクションを行っている．

3.2 PUD でのアルゴリズム実装

では，このリダクションを多数決演算ベースの PUD で実装するにはどうすればよいか．全加算器については，以下のよう 3 変数の多数決演算 (MAJ3) によって計算ができる．

$$C_{\text{out}} = \text{MAJ3}(A, B, C_{\text{in}})$$

$$S = \text{MAJ3}(A, \neg C_{\text{out}}, \text{MAJ3}(B, C, \neg C_{\text{out}}))$$

PUD では 5 変数等での多数決演算も可能であるが，精度を考慮しここでは MAJ3 のみを使用する．

全減算器についても，同様の MAJ3 の計算グラフで，オペランドの反転の有無のみを入れ替えたもので実現ができる．

$$C_{\text{out}} = \text{MAJ3}(\neg A, B, C_{\text{in}})$$

$$S = \text{MAJ3}(A, C_{\text{out}}, \text{MAJ3}(\neg B, \neg C, C_{\text{out}}))$$

これは非常に重要な性質であり，この性質によって後で述べるように同一のテンプレートを用いて全加算器と全減算器をバンク並列に実行することができるようになっている．

3.3 Booth エンコーディング

リダクションの回数は，ベクトル中に 1 が立っているビットが多いほど増加する傾向がある．そこで，そのような場合にリダクション回数を減らすために Booth エンコーディング [1] を採用する．これは冗長 2 進数の一種であり，各ビット位置

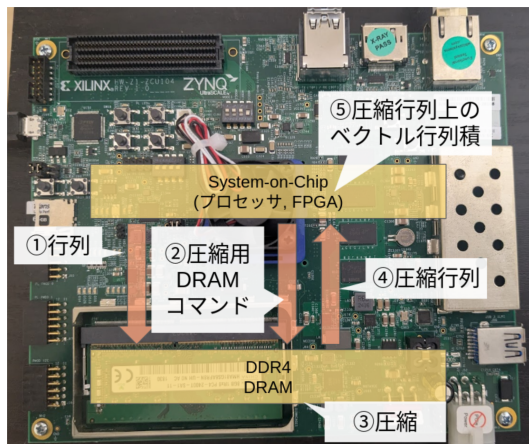


図2 システムの構成と実行フロー。

Fig. 2 Configuration of the system and execution flow.

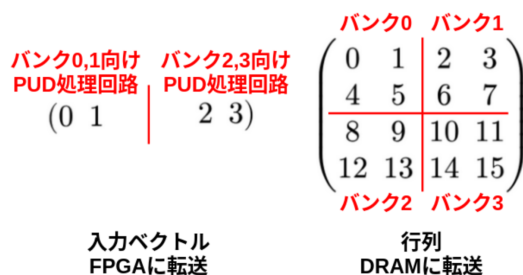


図3 行列・ベクトルの分割と転送先。

Fig. 3 Partitioning and transfer of matrix and vector.

に-1が現れることを許可する。7 = 111₍₂₎ のように1が連続する場合に 8 - 1 = 1000₍₂₎ - 1₍₂₎ と表現し、1が現れるビット数を減らすことができる。

Booth エンコーディングを採用することにより、リダクションの回数は単なるビット密度ではなく、「入力ベクトル中にビットがどれくらい乱雑に出現するか」いわばビットの複雑さに応じて増加するようになる。このことについては実験で検証する。

4. システムレベル実装

図2に示すように、本研究ではPUDに用いるDDR4 DRAM、FPGA ロジック、そしてSoCとして統合されたプロセッサが協調動作するシステムによって行列ベクトル積実行の高速化を行う。具体的には、DDR4 DRAMが行列を保持し、PUDによって行列のリダクションを実行する。FPGA ロジックには高位合成された高性能なPUD処理回路がDRAMのバンク数分展開されており、タスク並列性を実現している。プロセッサは行列やベクトルの転送と配置を担う。

システム全体としての動作を、図2のフローを元に以下で説明する。

4.1 行列の配置

まずDRAMに行列を配置する。転置はせず、行列の各行がDRAMのサブアレイ上の各行に配置されるようにする。従来のSIMDRAM等とは対照的に、行列の各要素の全てのビット

は同一の行に配置され、ビット位置ごとに並列に全加算器・全減算器のビットシリアル演算が適用されるようにする。ただし、PUDの制約のためDRAMには行列の各行とそのビット反転を配置しなければいけない。具体的には、行列の1行目がサブアレイの1行目、行列の1行目のビット反転がサブアレイの2行目、行列の2行目がサブアレイの3行目、…と配置される。

DRAMのバンク並列性を活用するため、行列は複数バンクに分散させることが望ましい。リダクションの適用可能性を維持するため、図3に示すように、行列はブロック状に分割し各バンクに配置する。

なお、大規模言語モデルの推論等、同一の行列を用いた行列ベクトル積を複数回実行する場合、行列の配置はプログラム開始時に一度のみ行えばよい。

入力ベクトルはPUD用のDRAMには書き込まないが、FPGA上のPUD処理回路は保持する必要がある。図3に示すように、各バンクが保持する部分行列の計算に必要な部分ベクトルを、各バンクに対応するPUD処理回路が保持するようにする。

4.2 リダクションのDRAMコマンド送信

各バンクごとのPUD処理回路が入力ベクトルをもとに、リダクションを独立・並列に実行する。実行には入力ベクトルの解析（次にどの行を用いてリダクションを行うかの決定）とDRAMコマンド発行が含まれるが、この2つはパイプライン化が可能である。DRAMコマンド発行がレイテンシを支配するよう、入力ベクトルの解析は高速に実行される。

発行されたDRAMコマンドはFPGA上に実装されたカスタムメモリコントローラーによってDDR4 DRAMに送信される。このメモリコントローラーはベンダー提供のDDR4 PHY IPを基に構築されている[13]。

4.3 リダクションのPUD実行

発行されたDRAMコマンドを受け、DRAM内部で超並列に計算が実行される。

4.4 圧縮行列の転送

リダクション実行後、圧縮された行列をDRAMからFPGA側に転送する。この処理はPUD実行と並列化はできないため、リダクションが完全に終わってから実行される。圧縮行列と、PUD処理回路が保持している圧縮ベクトルを用いた行列ベクトル積の計算はFPGA側で行われ、圧縮行列の読み取りとパイプライン実行される。パイプラインのレイテンシはやはりメモリアクセスに支配される。

転送されるのはサイズの小さい行列であり、かつリダクションは比較的小さなレイテンシでバンク並列に行うことができるため、元の行列を転送した際のメモリバンド幅を超えることができる。

5. MIMD バンク並列

4.1では、DRAMのバンク並列性を活用して計算の並列度を高めるために、行列を複数バンクに分散させた。これにより、異なるバンクが異なる入力ベクトルを処理することになる。動的PUDを用いた本手法ではDRAMコマンドが入力ベクトルに依存するため、これには「バンクごとに異なるPUD

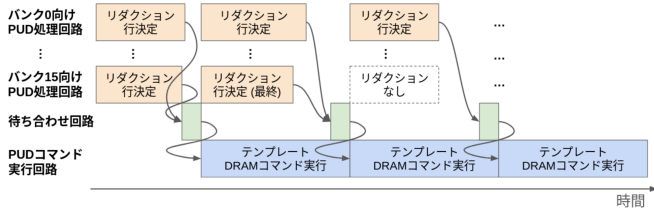


図4 バンクごとに異なるリダクションを行うことを想定したパイプライン設計.

Fig. 4 Pipeline for heterogeneous bank-parallel reduction.

コマンドを同時に発行する」という、MIMDのような状況が生じることになる。各バンクはDRAMとSoC間のチャネルを共有しているため、これに対処するにはDRAM上のタイミング制約やPUD実現のための条件を保ったままPUDコマンドを重ね合わせる必要がある。

本研究では、PUDコマンドの高スループット転送とバンク並列性の活用を両立させるために、**テンプレート方式**を採用する。これはバンク並列で高度に最適化されたDRAMコマンド列（テンプレート）を予め用意しておき、オペランドや少量の実行時のパラメーター（全加算器か全減算器か等）をバンクごとに注入して実行する方式である。3.2で述べたように、全減算器は全加算器と同様のDRAMコマンド列で、オペランドを適宜変更するのみで実現することができる。

図4にMIMDバンク並列性を考慮した実行パイプラインを示す。各バンクで独立に決定されたリダクション行（と、加えて全加算器を実行するか全減算器を実行するかの情報）が集められ、それがテンプレートに流し込まれ、コマンドが実行される。各バンクがリダクションする行を決定するタイミングは異なりうることを述べておく。そのため、各バンクのPUD処理回路からの出力はハンドシェイク信号を活用して待ち合わせをし、全ての出力が揃った段階でテンプレートに注入してPUDコマンドを実行する。これらの処理は全てパイプライン化され、やはりPUDコマンド実行が全体のレイテンシを支配するようになる。

なお、リダクション回数についてもバンクに応じて異なりうる。図4のバンク15での2回目のリダクションのように、各バンク最後のリダクションについてはフラグを立てておき、それ以降待ち合わせをしないようにする。テンプレート実行において、リダクションが終了したバンクはダミーのオペランドを指定することで副作用を防ぎつつコマンドを実行できる。

また、メモリの読み出しについてはバンク並列性を活用できないため、圧縮行列の転送に関しては待ち合わせ等は行わず、各バンク直列に処理する。

6. 評価

6.1 評価手法

AMD Zynq UltraScale+ MPSoC ZCU104の開発用ボードを用いて、Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoCと16GB DDR4 SODIMM HMA82GS6AFR8N-UHを接続した。高位合成デザインはVitis 2024.2で合成し、ハードウェアデザインは

表1 実験結果の概要.

Table 1 Summary of results.

アルゴリズム	平均スループット [GB/s]	平均 NMSE (符号付き入力)	平均 NMSE (符号なし入力)
MVDRAM	16.87	0.430	0.123
MVDRAM.B.	19.03	0.430	0.119
PATTERN	21.54	0.514	0.094
PATTERN.B.	22.55	0.554	0.087

Vivado 2024.2で合成した。PSのソフトウェアはPYNQ Linux Image v3.1.1 for ZCU104上で動作した。

行列は8bit、ベクトルは4bitで量子化した。行列のサイズは、ここでは各バンク1サブアレイで16バンクを使い切る大きさにした。具体的には、1サブアレイに 128×8192 の部分行列を配置し、入力次元 128×8 、出力次元 8192×2 の行列とした。

ここでは、メモリバンド幅との比較を明らかにするため、計算スループット、具体的には1秒間にどれくらいの量の行列データを処理できるかをを用いて計算性能を表すことにした。また、DRAM計算は誤差を伴うため、計算精度については計算結果に対して以下で計算されるNMSE (Normalized Mean Square Error) をを用いて評価した。

$$NMSE = \frac{\|\mathbf{v}_{PUD} - \mathbf{v}_{ideal}\|^2}{\|\mathbf{v}_{ideal}\|^2}$$

実験結果は、リダクション方式やBoothエンコーディングの使用有無の条件を変更して計測した。先行研究のMVDRAM[7]で提案された、ベクトルの各値をビットごとに分解してリダクションする手法をMVDRAM、それにBoothエンコーディングを使用したものをMVDRAM.BOOTH、本研究でのパターンマッチの手法をPATTERN、それにBoothエンコーディングを適用したものをPATTERN.BOOTHとして評価した。リダクションの回数はビットパターンに依存するため、ここではビット密度を変化させてランダムに生成した入力ベクトル値を元に評価した。

6.2 メモリバンド幅の計算

ZCU104でサポートされており、かつDDR4で一般的なチャネルあたりの転送速度である2133MT/s (mega-transfers per second) を元にメモリバンド幅の算出を行った。

$$\text{Bandwidth} = 2133 \text{ MT/s} \times 8 \text{ Bytes/Transfer} = 17.064 \text{ GB/s}$$

6.3 結果

表1に実験結果の概要を示す。MVDRAM以外のアルゴリズムにおいて、ビット密度を変化させた場合の平均スループットはメモリバンド幅を超えることができた。パターンマッチとBoothエンコーディングの両方を適用したPATTERN.BOOTHアルゴリズムが最大の平均スループットを達成した。NMSEについては、入力値が符号なしの場合は0.1程度となったが、符号ありの場合は0.5程度と大きくなった。これは引き算が発生することでNMSEの分母のノルムが小さくなったためであ

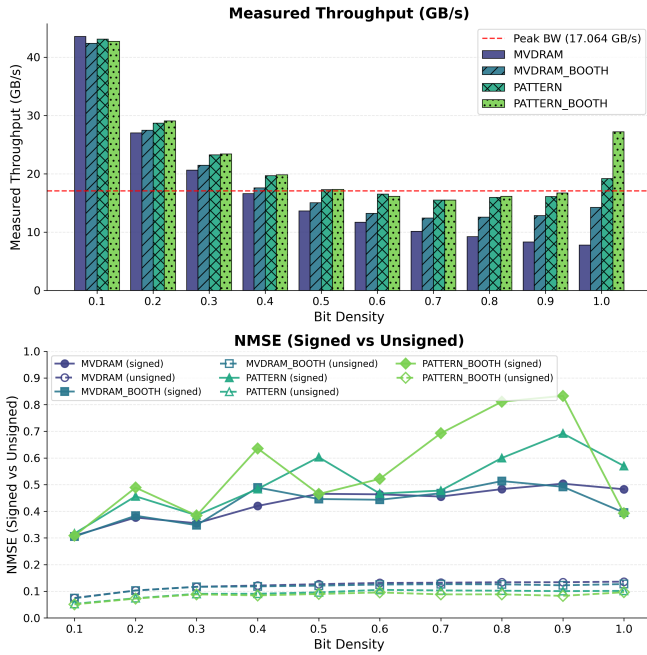


図5 ビット密度ごとのスループットと NMSE の比較.

Fig. 5 Comparison of throughput and NMSE for each bit density.

と考えられる。符号なしの場合は PATTERN_BOOTH アルゴリズムの NMSE が最小となった。リダクション回数が少ないとスループットの上昇と精度の改善が両立するためであろう。

図 5 にビット密度を変化させた場合の計算スループットと NMSE を示す。計算スループットについては 10 個中 8 個のビット密度で PATTERN_BOOTH アルゴリズムが最大の性能を発揮した。MVDram アルゴリズムがビット密度の増加に応じて単調に性能が低下したのに対し、特にパターンマッチによるアルゴリズムは下に凸な結果を示した。また、凸性はビット密度が高い方に歪み、密度 0.7 が最も低い性能を示した。これは、ビット中の 1 の割合が 1.0 に近づくにつれビットパターンは単純に近づくといえるが、それでも 0 が多数の場合よりリダクションが必要であるからだと説明できる。0 が多数の場合に性能向上が大きい点はビットのスパース性を活用するアーキテクチャとして期待できるかもしれない。一方で、ビット密度が 0.6 から 0.9 の範囲では性能はメモリバンド幅を超えられなかった。これは今後 PUD の動作周波数を最適化すること等により改善できると考えられる。NMSE については入力値が符号なしの場合ではビット密度による影響は少なく安定した結果となったが、符号付きの場合はビット密度やアルゴリズムにより大きく変動した。

7. おわりに

本研究では PUD を FPGA と協調させて行列ベクトル積を高速化し、実ハードウェアでメモリバンド幅を超える計算スループットを検証した。今後は PUD 動作周波数の最適化により計算スループットと演算誤差を改善し、また実アプリケーションに PUD 計算が統合されるシステムの構築に努めることが課題となる。

謝 辞

本研究の一部は、JST CREST JPMJCR21D2 および JSPS 科研費 23H00467 の支援による。

文 献

- [1] Andrew D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, Vol. 4, No. 2, pp. 236–240, 1951.
- [2] Robert H Dennard. Field-effect transistor memory, June 4 1968. US Patent 3387286A.
- [3] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. Computedram: In-memory compute using off-the-shelf drams. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pp. 100–113, 2019.
- [4] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system. *IEEE Access*, Vol. 10, pp. 52565–52608, 2022.
- [5] Nastaran Hajinazar, Geraldo F Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. Simdram: A framework for bit-serial simd processing using dram. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 329–345, 2021.
- [6] JEDEC Solid State Technology Association. *DDR4 SDRAM STANDARD*. JEDEC Solid State Technology Association, 2021. Accessed: 2026-4-27.
- [7] Tatsuya Kubo, Daichi Tokuda, Tomoya Nagatani, Masayuki Usui, Lei Qu, Ting Cao, and Shinya Takamaeda-Yamazaki. Mvdram: Enabling gemv execution in unmodified dram for low-bit llm acceleration. *arXiv preprint arXiv:2503.23817*, 2025.
- [8] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyoungwan Lim, Hyunsung Shin, et al. Hardware architecture and software stack for pim based on commercial dram technology: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 43–56. IEEE, 2021.
- [9] Jiantao Liu, Minxuan Zhou, Yue Pan, Chien-Yi Yang, Lana Josipović, and Tajana Rosing. Optimim: Optimizing processing-in-memory acceleration using integer linear programming. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 867–883, 2025.
- [10] Onur Mutlu, Ataberk Olgun, Geraldo F Oliveira, and Ismail E Yuksel. Memory-centric computing: Recent advances in processing-in-dram. In *2024 IEEE International Electron Devices Meeting (IEDM)*, pp. 1–4. IEEE, 2024.
- [11] Geraldo F Oliveira, Ataberk Olgun, Abdullah Giray Yağlıkçı, F Nisa Bostancı, Juan Gómez-Luna, Saugata Ghose, and Onur Mutlu. Mimdram: An end-to-end processing-using-dram system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 186–203. IEEE, 2024.
- [12] İsmail Emir Yüksel, Yahya Can Tuğrul, F Nisa Bostancı, Geraldo F Oliveira, A Giray Yağlıkçı, Ataberk Olgun, Melina Soysal, Haocong Luo, Juan Gómez-Luna, Mohammad Sadrosadati, et al. Simultaneous many-row activation in off-the-shelf dram chips: Experimental characterization and analysis. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 99–114. IEEE, 2024.
- [13] 久保龍哉, 勝見舜, 篤田大知, 三好健文, 高前田伸也. 直接的に記述可能な低レベル dram インターフェースの開発. 情報処理学会研究報告 (IPSI Technical Report), Vol. 2026-ARC-264, No. 20, pp. 1–9, March 2026. Web Only.