Vol.2022-ARC-248 No.10
Vol.2022-SLDM-198 No.10
Vol.2022-EMB-59 No.10
2022/3/10

汎用粗粒度再構成可能アーキテクチャの検討

高野 茂幸[,a)]　天野 英晴[1,b)]

概要：深層学習などの用途についてデータフロー型計算が再認識されている．近年, 特定用途向けのアクセラレータが注目されているが, 本研究では汎用とした場合の CGRA を検討し, 今回ストリームデータフロー型計算向けアーキテクチャについてそのベースラインを高水準ハードウェア開発言語である Chisel で PoC (Ploof of Concept) を開発したのでアーキテクチャの概要を説明した上で初期評価の結果を報告する．

1. はじめに

近年 CGRA(Coarse-Grained Reconfigurable Array; 粗粒度再構成可能アレイ) アーキテクチャが再認識されている．制御フローが不要なデータフローグラフとして扱える深層学習などのアプリーケーションが一般化し, それが数値演算を伴うデータフローグラフで構成できる CGRA に向いているからである．現在シリコンリソースは潤沢にあり, またアプリケーションは大量のデータを伴い演算への要求も高い．本研究では従来の汎用プロセッサが行なっている多大なリソースを費やして独立に実行可能な命令を見つけて実行するのとは対照的にデータ依存性を利用してデータ依存性グラフをチップ上に構成して, そのグラフの持つデータレベル並列性を最大限活かす方向性を検討している．

一方で深層学習の分野では特定用途向けアクセラレータ (DSA; Domain-Specific Accelerator) も積極的に研究され提案されている [2][3][4][12][14]．BLAS ライブラリに向けたアクセラレータとして行列積に特化したものを中心に, 畳み込みニューラルネットワークに特化したアクセラレータの研究が盛んに行われている [2][3][4][12]．この分野でのアクセラレータの基本構成として比較的大きいバッファを一つ用意しておき, 外部メモリへのアクセスを削減する事を図っている [12][14]．外部メモリアクセスは一般的に数十から数百クロックサイクルを伴うので遅延の原因であり, またこの外部メモリへのアクセスが一番エネルギーを消費するため, 高い実行性能と低い消費エネルギーを実現する上で鍵となっているからである．

深層学習のモデル研究は日進月歩であり, 様々なネットワークモデルが提案され, 特定のモデルに注目されると今まで使用されていたモデルが使われなくなる事がしばしばある．従って, 例えば畳み込みニューラルネットワークについて特定のパラメータを持つネットワークモデルに特化してしまうとそのハードウェアは利用出来なくなる事が発生しやすい．畳み込み演算自体が入れ替わりの対象になり得る [1]．産業の観点では DSA を俯瞰すると上記のようにアプリケーションの変化が激しいと半導体チップとして製造した場合, 少量生産になり収益を得にくくリスクが大きい．従って, FPGA(Field-Programmable Gata Array) といった再構成可能なアーキテクチャ上に実装する事が現実解になる．しかし, FPGA は単ビット出力である Look-Up Table を計算ノードとしており, 数値演算を伴うとリソース使用とクロックサイクル時間の点でメリットを享受できない．そこで数値演算ノードを使用した CGRA が再認識され始めており製品としてリリースされたり [5][6], また研究が活発になりつつある [15][16][17].

DSA のアプリケーションの急激な変化に対応できない短所を克服しつつ従来のプロセッサの様な汎用性を持つアーキテクチャの一つの方向としてホストシステムを持たない自立再構成が可能な CGRA アーキテクチャ, ElectronNest(EN) を本論文で提案する．従来のプロセッサが汎用である所以はこの制御フローの切り替えに対応して柔軟なプログラムの開発を実現している点に尽きる．実現するには制御フローの取り扱いを検討する必要があるので, 今回ベースラインの EN を開発した．その機能を中心に動作を検証し初期評価を行ったので結果を報告する．

次の章では汎用 CGRA アーキテクチャを検討するにあたり, どのような点に着目しているかを説明する．それに対してどのようなアプローチを取るかを第 3 章で説明して

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，慶應義塾大学
Presently with Keio University
[a)] takano@am.ics.keio.ac.jp
[b)] amano@am.ics.keio.ac.jp

Vol.2022-ARC-248 No.10
Vol.2022-SLDM-198 No.10
Vol.2022-EMB-59 No.10
2022/3/10

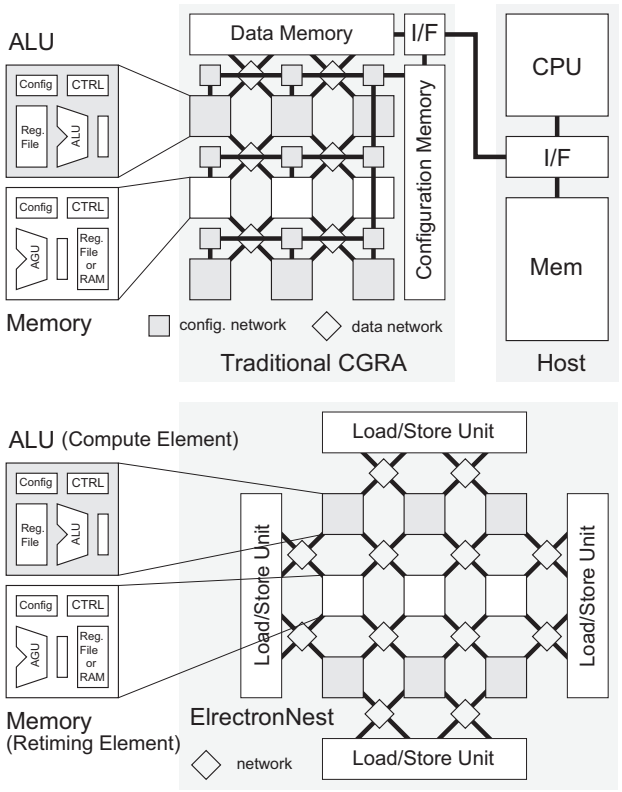

図 1 CGRA と ElectronNest の違い

いる．第 4 章では基本動作について例を用いて説明している．第 5 章ではその初期評価結果を説明し, 最後の章でまとめている．

2. 汎用 CGRA アーキテクチャの課題

この章では提案する EN のアーキテクチャを説明する前に汎用性のある CGRA を検討する際の課題を取り上げる．図 1 上下はそれぞれ一般的な CGRA とそれに対応する EN の基本構成を示している．一般的な CGRA ではホストシステムに接続しており, ホスト側が CGRA の構成やデータを制御する．これに対して EN では簡素な構成を持ち, 自律動的再構成を行う．ENA は演算を行う ALU ユニット (CE; Compute Element) とデータを保持するユニット (RE; Retiming Eelement) の 2 種類で構成されている．外部メモリに接続されているロードストアユニットのアーキテクチャは RE と同様でメモリ本体を除いたものである．

2.1 汎用性の定義と制御フロー

CGRA アーキテクチャを検討するにあたり, 汎用 CGRA とは一般的なソフトウェア・プログラムを対象として, 数値演算処理を少ないオーバヘッドで行えかつ制御フローを容易に扱えるアーキテクチャとする．CGRA の汎用化にはプロセッサの様に様々なアプリケーションに対応する必要があり, 制御フローをどう扱うかを検討する必要がある．

ソフトウェア・プログラムの基本ブロックをデータフローグラフとして, 特に頻繁に使用される部分を選びそのフローグラフを CGRA 上に構成させるのが一般的である．FPGA や CGRA での制御フローに対してのスケジューラは幾つか提案されているが, そのほとんどはコンパイル時に静的に決まるものを対象としている．深層学習もその例であり, 例えば畳み込み演算のネストされたループ構造を FPGA や ASIC として実装しており, 制御フローを回路中に内包させてフローの分岐を削除している．

静的にユーザー回路を構成する CGRA では, 例えばホスト・プロセッサにより CGRA 上に構成するユーザー回路のスケジューリングや構成と実行の管理を行う方法が主流で自律再構成なアプローチは少ない [7][11]．一般に FPGA に見られるように一度チップ上に構成した後は構成された同じ演算を実行し続ける事を前提としている．従って, 最も負荷の大きい部分をタスクとして元のプログラムから分割してハードウェアとして設計して CGRA や FPGA 上に構成するのが一般的な手法である [7][10][11]．つまりユーザー回路構成間の状態遷移はなく, 実行時の状況に応じて再構成する事が難しい．複数の基本ブロックに対してその選択を行う場合, ユーザー回路間に存在する制御フローの分岐に対応する必要がある．

2.2 ルーティングと構成方法

従来, データと構成データの転送にそれぞれ別のネットワークを用意しておき, 実行のバックグラウンドで構成データを送信し構成のオーバヘッドを隠蔽する事が試みられてきた (図 1 参照)[7]．他方で, ルーティングしながらデータフローグラフを構成しつつ, 演算も行う方法も提案されている [9]．

前者の方法は演算と構成のスケジューリングを明示的に分割するので制御や管理が容易になる．その代償としてそれぞれに対してインターコネクトを用意し, また並行して使用するために構成データとデータそれぞれについてメモリを用意する必要もある．また, 静的に構成して同じ演算を繰り返すのであれば構成に要するリソースの使用は効率的と言えない．後者の方法はインターコネクトをデータと構成両方で共用するのでリソース使用を軽減できるがそれぞれについて制御や管理が不明瞭になり, 何らかのルールを設計する必要がある．

3. EN アーキテクチャの検討

ElectronNest ではオンチップメモリと計算ノードが分散にかつ特定パターンで配置されている．図 1 に示す様にその周辺にロードストアユニットが配置されており, このユニットを介して外部メモリへアクセスする構成をとる．

3.1 プログラミングモデル

本研究ではループ処理のインデックス情報を解析し, データ依存性グラフを構築して, このグラフを構成データとし

Vol.2022-ARC-248 No.10
Vol.2022-SLDM-198 No.10
Vol.2022-EMB-59 No.10
2022/3/10

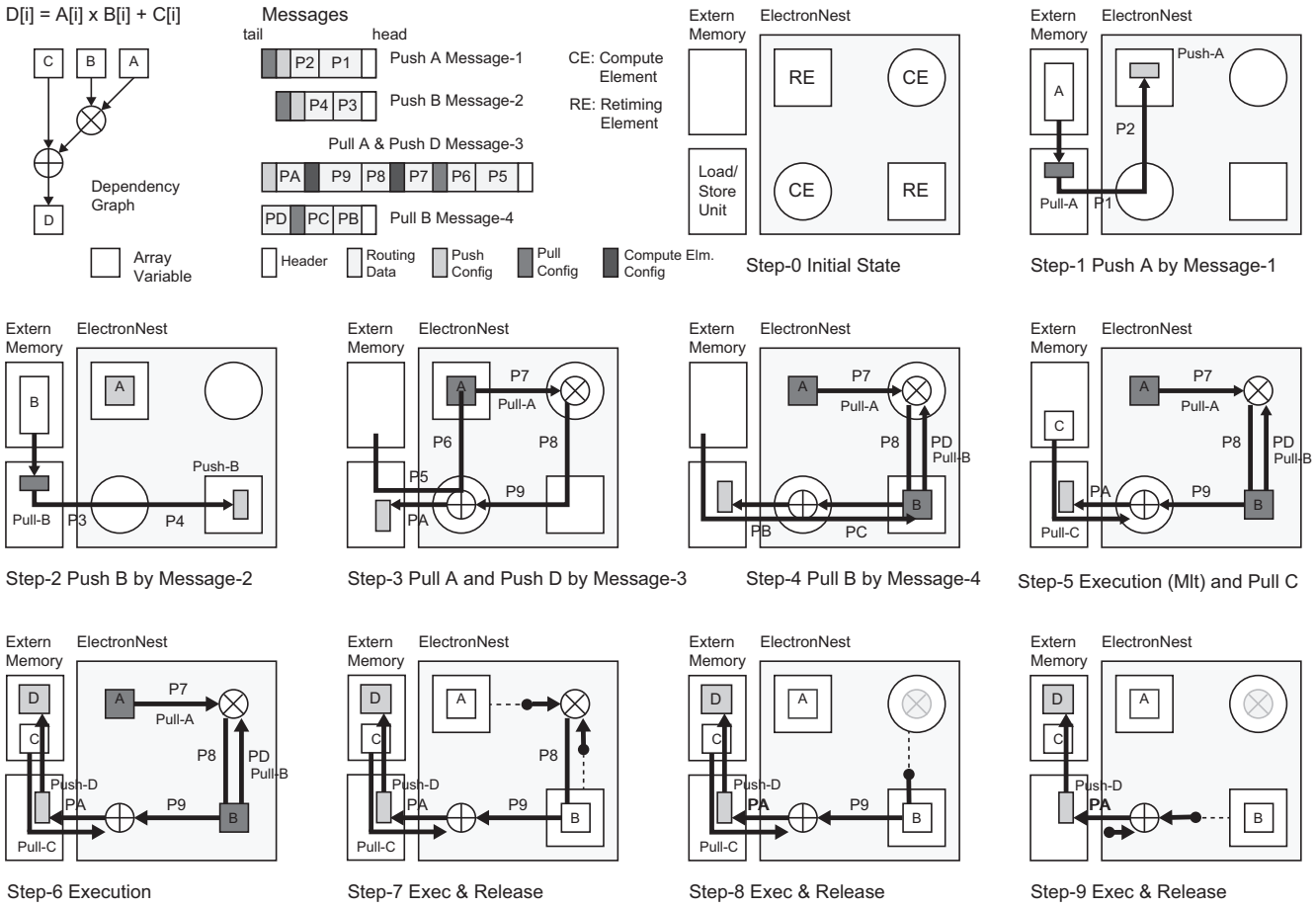

図 2 メッセージとそれによるユーザ回路の構成, 実行そして解放

て CGRA へマップする事を狙う．つまり, 繰り返し間にデータ依存性があってもループ展開してユーザ回路を構成できる様にする事を目指している．図 2 左上はベクトル MAD(Maltiply-Add) のデータ依存性グラフを示している．丸は演算ノード, 四角は配列である．この場合イテレーションの間にデータ依存はないので SIMD の様なデータ並列に演算が可能であるが, 簡易に説明するために例として用いている．グラフの右側にベクトル MAD に対応する後述するいくつかのメッセージを示している．

EN では構成データを含むすべてのデータをメッセージとして扱う．メモリへのロード・ストアはメッセージを単位として行う．また, メッサージの基本的な内容は, ヘッダ, ルーティングデータ, 構成データ, そして演算に用いるデータである (図 2 参照)．メッセージは EN の Element アレイ上をワームホール・ルーティングで移動する．図 2 上真中のメッセージ中に”P”とラベルされているデータはルーティングデータであり, メッセージの移動の経路を設定する際に使用され, ステップ 0 からステップ 9 で示す様な実行時にラベルされている経路と一致している．ルートを選択するために一つのルーティングデータを消費する．消費したルーティングデータはメッセージから削除されるので移動と共にメッセージは短くなる．ヘッダ以外の他のデータも消費されればメッセージから削除される．例えばステップ 1 では外部メモリにある配列変数 A を EN の Retiming Element へストアする事を試みている．メッセージ 1 のルーティングデータ P1 と P2 でルーティングを行うが, ルーティングデータ P1 を使用して経路を確立した後, このルーティングデータ P1 はメッセージ 1 から削除される．

演算に用いるデータは外部メモリ, 或はオンチップメモリに予めストアされている事を前提としている．EN では標的の Element へデータをストアする事をプッシュと呼んでいる．逆に標的の Element からデータをロードする事をプッシュと呼んでいる．EN はプルとプッシュの組み合わせで動作する．例えると RTL 言語のレジスタからレジスタへの転送をこのプッシュ・プルで実現している．つまり, データフローの始端と終端を指定する．メッセージ 1 と 2 はそれぞれ配列変数 A と B を Retiming Element へプッシュするが, これらのメッセージは配列変数を持たない．ロードストアユニットにおいてメッセージの終端で外部メモリに対してプルを行い配列変数を読み出す．メッセージからデータ本体を分離しており, 従来のプロセッサ同様にプログラムとデータを分離して扱う事に相当する．Step-3 から Step-4 への遷移で示す様に配列変数 A につい

Vol.2022-ARC-248 No.10
Vol.2022-SLDM-198 No.10
Vol.2022-EMB-59 No.10
2022/3/10

て P7 以降の経路は後述する解放トークンが発火しない限り維持される．配列変数 B と C も同様である．Step-5 で示す様にユーザ回路の構成と乗算の実行はオーバラップされており, 後続の経路が確立待ちになるまで演算を行い結果を出力する．

3.2 制御フロー

メッセージの選択を条件により選択可能にする事でユーザ回路の構成を条件付きで可能にしている．図 3 は制御フローを導入する際の各所の構成を示している．図 3 における Load Unit は図 1 におけるロードストアユニット内にある．その Load Unit の基本構成を図 3 左上で示している．図 3 右上での丸印は乗算と加算の演算ノードであり図 1 の Compute Element に配置される．四角印は配列データで, Retiming Element に配置される．このデータ依存グラフを図 3 下の EN 上に構成する例で説明する．図では省略しているが Compute Element や Retiming Element の間にはルータがあり, パイプライン動作する．

右上に構成データとなるデータ依存性グラフを 2 つあり, これらを図 3 左下の様に外部メモリのストアしてあるとする (Config.1 と 2)．これらは前の章で示したメッセージの集合である．

図 3 に示すように制御フローに関して, パイプライン上に比較結果といった条件信号 (condition) をデータの流れと逆方向に伝播させて, 入力するデータを選択可能にしている．例えば加算を行う Compute Element の演算結果をゼロ比較してその結果を条件信号として前方向と逆方向に伝播させる事ができる．条件信号の逆伝播によりデータ依存性グラフの始端までこの信号が来ればメモリアクセスのロード（プル）を条件付きで行えるようになるので, 次に構成するユーザ回路 (メッセージ) が選択可能になる．メッセージ単位でメモリからロードしていくが, そのロードの終了時の条件でメモリアドレスのオフセット値や絶対値を選択し, 次の構成ブロックへ分岐する (図 3 左上参照)．図 3 は分岐先を 2 つとしているが, 例えば条件が真と為の 2 つのオフセット値を持つこともできる．このためにも, データ転送と構成データ転送を同じインターコネクトで行う事によりアーキテクチャを簡素なものにしている．

従来のソフトウェア・プログラムでは if 文による条件付き代入で発生する ϕ がある．ソフトウェア・プログラム上 if と else それぞれで同じ変数へ条件に基づき代入するとする場合, if と else それぞれの基本ブロックのどちらかの値を代入する事になる．この時の入力条件を伴う分岐を一般に ϕ と呼んでいる．ベースラインでは演算回路間の経路を実行中静的なものとして EN 上にユーザ回路を構成し図 3 右下の様に ϕ を実現し, また条件付きメモリアクセスで構成データも含めたデータの流れを変える．その拡張としてオンチップルータ上で条件により到着している複数データ

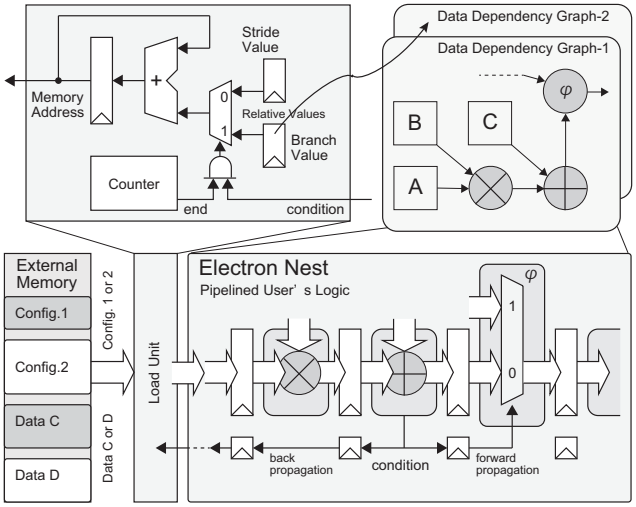

図 3 CGRA への制御フローの導入

の中から出力するデータを選択できるようにして後で説明する様にデータフローの順番を保証している．逆伝播では解放トークンをイベントとして送信し条件のトリガとしているが, ϕ では逐次条件を利用する．

3.3 ルーティングと構成方法

EN ではオンチップ・ルータをユーザ回路の構成に用いるアプローチを取り, シリコンリソースを演算とそのためのインターコネクトへ最大限充てる事を狙う．つまり, 図 2 に示した様にルーティングにより演算回路間の経路を構成し最終的にユーザ回路のデータ依存性グラフを構成させる．そのために, 一度構成した経路を保持し, また最後に解放する必要がある．これを実現するために解放トークンを用意した．

図 2 の黒ドットはつ解放トークを持つ最後のデータを示している．解放トークンは最後のデータに付与される．経路上で解放トークンを検出した時その経路を構成しているルータは解放される．演算回路では各ソースオペランドが持つ解放トークンが発火した後にその演算回路を解放する．従って, 構成したデータ依存性グラフの経路に従って順次使用しているリソースを解放する．

データにはヘッダ情報が付与されており, 3 つのデータで構成されている．データ自身の ID 情報一つとルータが選択すべき次のデータの ID 情報二つであり, 条件信号により二つの内いずれかを選択する．到着した複数のデータの内 ID が一致しているデータを選択してルーティングを行う．これにより使用する構成データやデータの順番を保証しながらデータフローを条件付きで制御できるようにしている．

4. 汎用 CGRA のプロトタイピング結果

汎用 CGRA アーキテクチャを検討するにあたり, 検証を目的とした PoC を開発している．今回ベースラインの開

発が終わったのでこの章ではその基本動作を説明する．

4.1 開発環境

開発にあたり Chisel 言語を使用している．Chisel 言語はソフトウェア開発言語である Scala 言語を拡張したものであり, Scala 言語の文法に従った記述が可能である．Chisel 言語で記述されたハードウェアは tester2 と呼ばれるテストツールを用いてシミュレーション検証ができ, VCD ファイルを生成できるので波形を確認する事ができる．また, Verilog-HDL ファイルを生成できる．これにより FPGA 論理合成と配置配線が可能であり, 初期動作検討が可能である。

今回のプロトタイプ開発では Chisel の特徴であるパラメタライズ記述を利用して開発したので, データ幅, アレイサイズ, チャネル数, Compute Element 内 ALU 数, ALU 構成, Retiming Element 内 RAM 数, RAM サイズなどを再定義可能である．ただ, アレイのネットワークトポロジは直接ポートを繋ぐ必要があるので現状パラメータ定義できていない．

今回データ幅は 32 ビットワードとしており, 外部メモリアクセスのインターフェイスも同じ幅に設定している．

4.2 動作検証

図 4 は 2 x 2 のアレイと外部メモリへ接続しているロードストアユニットで構成された EN のシミュレーション結果を示している．アレイは Compute Element と Retiming Element の 2 種類でそれぞれ構成されており, 動作は図 2 である．この例では D[i] = A[i]×B[i]+C[i] を計算している $(0 \leq i \leq 255)$.

標的の Retiming Element まで来たらデータをストア (St_Data_D[31:0]) している．A と B に対してこれを行う．続けて配列変数 A に格納している RAM まで行き, データをプルしつつ続けて Compute Element 内で演算回路を構成, さらに次の Compute Element の入力まで経路を確立する．配列変数 B は初めの Compute Element まで同様に行う．ルーティングで確保した経路はパイプライン動作する．ソースオペランドが揃うようにタイミング調整のためにパイプラインレジスタを挿入したり経路長を調整する必要がない．

Compute Element(A×B) は A と B の要素の到着と共に演算を開始して結果要素を次の Compute Element まで構成した経路で転送する．C のプルと Compute Element への経路の接続により, 加算を開始する．全てのオペランドの到着と共に演算し, 到着しない場合に nack を逆伝播させて同期を取る (上図の丸で囲った“Synch by nack”部分)．先行する乗算は C の要素データの到着まで nack によりストール状態にある．

A と B の終端のデータは解放トークンを持ち, 経路がデータフローに従って解放されていく．初めの Compute Element 内の演算器が解放トークンの発火で解放され, 次々に解放されていく．C も同様である．経路は解放トークンで解放されるのでこのトークンが発火しない限り維持される．従って経路の始端にデータを置けば終端までそのデータは自然に流れる．経路途中に演算ノードがあれば自律的にソースオペランドは同期を取り演算を行い, 演算結果を出力してそのデータが後続の経路を流れる．

5. 初期評価

初期評価として Cray 方式のベクトル演算について実施した．評価ではベクトル加算とベクトル乗和 (MAD) に対して行なった．

5.1 ベクトル加算

外部メモリアクセス用のロード・ストアユニットはロードとストア両方のインターフェイスを持つ．ベクトル加算は一つの Compute Element を使用して一方のソースデータを隣接する Retiming Element に予めストアしておき, その Retiming Element からのロード (Pull), Compute Element までのルーティングと内部の構成, 外部インターフェイスまでのルーティングとストアアクセスのための設定を行う．他方のソースオペランドは外部メモリからロード (Pull), Compute Element までのルーティングを行う．Compute Element でデータが揃ったところで演算を開始する．演算結果はストアまでの経路を流れ最終的に外部メモリへストアされる．この時, 630 クロックサイクル要している．また, 外部メモリ用ロード・ストアユニットがロードとストアどちらか一方のみ使用できる場合も評価した．この場合, 両方のデータを Retiming Element それぞれへ予めストアし, それぞれロード (Pull) している．この時は 917 クロックサイクル要している．

5.2 ベクトル MAD

ベクトル MAD については先の章で動作説明したものと同じである．先行するパイプラインステージで演算の同期を取れば Nack が逆伝播してパイプラインレジスタを順次ストールさせる．この時は 945 クロックサイクル要している．理想状態では 768 クロックサイクルなので, 23%のオーバヘッドであった．20 クロックサイクルが同期のための Nack の逆伝播により加算データのロード時にストールしており, 23%のオーバヘッドの内 3%を占めていた．残りの 20%はブート, ヘッダ, ロード構成データ, ルーティングデータなどであり, この内ルーティングデータは冗長データで, 28 クロックサイクルを消費している．これは 23%の内の 3.5%程度を占めている．今回の評価ではアレイサイズがごく小規模のため, 冗長なルーティングデータの影響は少なかったが, スケールした場合, 無視できない程度にな

Vol.2022-ARC-248 No.10
Vol.2022-SLDM-198 No.10
Vol.2022-EMB-59 No.10
2022/3/10

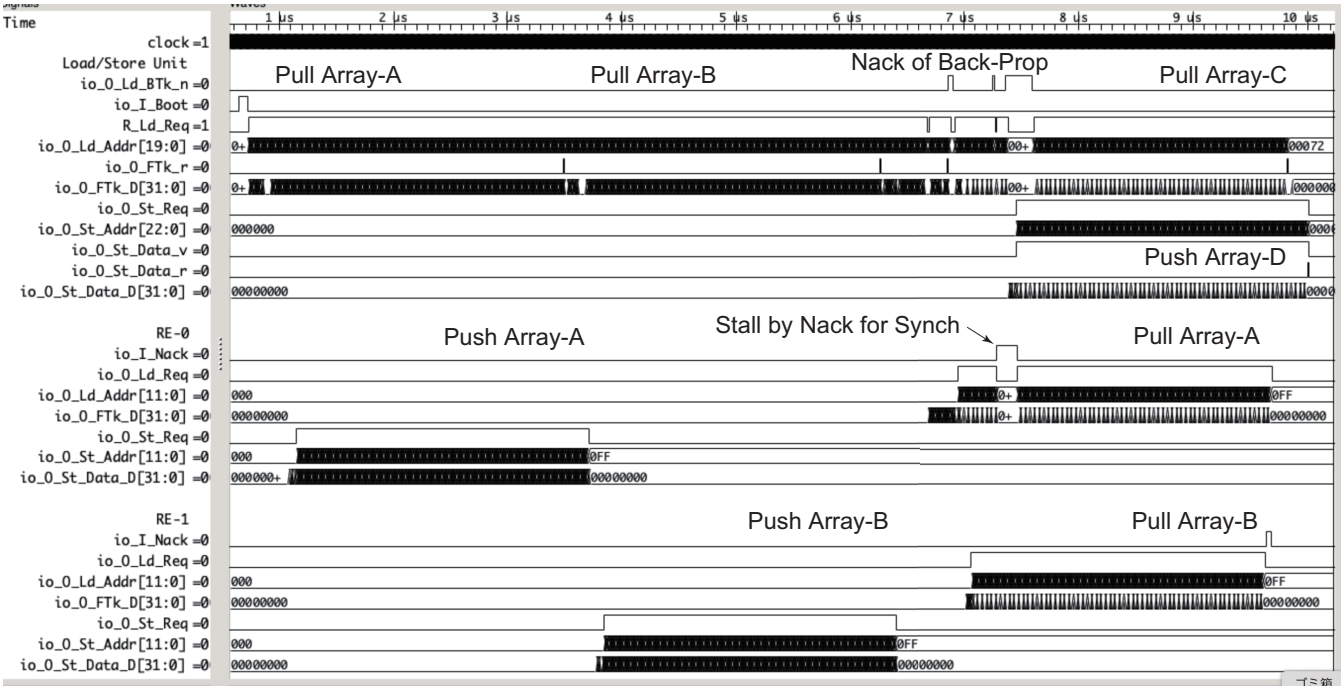

図 4 ElectronNest の Vector-MAD 動作例

る可能性がある．従ってルーティングデータの圧縮を検討する必要がある．残りの 16%程度のオーバヘッドはパイプラインで構成されているインターコネクト上でルーティングやデータ転送に要しているオーバヘッドであり, 削減は難しい.

外部メモリアクセスが明らかにボトルネックであり, これは複数のインターフェイスを採用する事により緩和されると推定される.

6. 関連研究

シストリック・アレイはプログラム中のループ部分に適用され, 同じ演算を繰り返す代わりにループ展開を行いチップ上にデータ依存性グラフを実装するアプローチである [13]．ループ部分は一般に配列を用いた演算を行う．プログラム上インデックスを用いて配列要素へアクセスする．このインデックスの取り方から配列の参照間隔を解析してループ展開へ反映させている．また, 真のデータ依存性を用いてループ繰り返し間のデータ参照パターンに従ったインターコネクトを構成する．Google 社が自社クラウド向けに内製したアクセラレータチップは行列積に特化したシストリック・アレイである [14]．行列積以外は出来ない.

カイザースラウテルン大学の Xputer プロジェクトは 90 年代初頭の CGRA 黎明期のプロジェクトであるが, 基本的な構成はこの時期に検討されていた [7]．アレイのノードは算術演算や論理演算が可能であり, ごく小規模のレジスタファイルも用意されている．アレイのノードの間は非同期ビットシリアル・インターフェイスである．画像処理のような二次元メモリアクセスに対応するため二次元のアドレスを従来のリニアなメモリ・アドレスへ変換するアーキテクチャも検討している [8]．また, 畳み込み演算に見られるストライド・ウィンドウのメモリアクセスを効率的に行うためにウィンドウ上のグリッドに対応したロード・ストアを明示する事によりストアからロードへのバイパスを行える仕組みも用意しており, データの再利用効率を向上させている.

カーネギーメロン大学の PipeRench プロジェクトも黎明期の CGRA の研究であり, ユーザ回路の構成方法に特徴がある [10]．ユーザ回路についてストライド・ウィンドウ内に該当する部分の構成データをパイプライン・レジスタを通して伝搬させ回路の構成とユーザ回路の実行を同時に行うことを狙っている．つまり, パイプラインで構成されたユーザ回路を仮想化して演算回路リソース以上の規模に対応させている．ウィンドウから出た部分の回路が持つデータはバッファにストアされ, 再びウィンドウに入ったにリカバリされる.

ウィスコンシン大学の DySER プロジェクトは汎用 CGRA のプロジェクトである [15]．CGRA 部を従来のマイクロプロセッサ・パイプラインに組み込んでいる．NoC を経由して演算ユニットがアレイ状に配置されている．プロセッサから CGRA を制御する命令数は 10 に満たない．別のプロジェクトでは DySER プロジェクトを基に CGRA のアレイ上にマッピングするルーティングを検討している．データを揃えて演算時の同期を取るためにタイミング調整を行うパイプライン・レジスタの挿入とバッファ

との組み合わせ, 経路長を揃える方法を検討している [16].

スタンフォード大学の Plasticine プロジェクトも汎用 CGRA のプロジェクトである [17]. 分散メモリを採用しており NoC を経由して隣接する演算ユニットへデータを渡す構成を採用している. 演算ユニットは SIMD 構成であり, 複数の演算回路による多段パイプラインとしている. メモリユニットはアドレス計算や前処理用の多段パイプライン構成の演算回路を経由して SRAM メモリへアクセスする. 論文中, NoC の構成に触れていないが NoC 間の演算ユニットやメモリユニットはパイプライン動作を行うが, データが揃ったところで実行を開始しているか, またパイプライン制御をどのように行なっているかは不明である.

7. まとめ

深層学習などのグラフで表現できるアプリケーションの登場により CGRA アーキテクチャが再認識され始めている. 深層学習タスクは静的なトポロジーを扱うので CGRA に向いたアプリケーションである. DSA を深層学習へ適用する研究は多くあるが, 深層学習タスクは日進月歩であり, タスクを構成する演算カーネルも栄枯盛衰の状況にある. 従って, 可能な限り汎用性を持たせた CGRA が深層学習タスクだけについても必要であり, また汎用化により適用できる分野が広がり低コスト化を踏まえて産業を考慮したとしてその利用・応用の面で寄与できると考えている. 本論文では汎用向けの CGRA に必要な課題について検討した上でアーキテクチャ, ElectronNest を開発し, そのベースラインの動作について説明した上で制御フローの扱いについて説明・提案した. そして高水準ハードウェア開発言語である Chisel 言語で PoC を開発し, 初期評価を行なったのでその報告をした.

参考文献

[1] Zhouhan Lin, Roland Memisevic and Kishore Reddy Konda：*How far can we go without convolution: Improving fully-connected networks*, The Sixth International Conference on Learning Representations (ICLR), Open Review(2018)

[2] Tianshi Chen, Zidong Du, Ninghui Sunand, Jia Wang, Chengyong Wu, Yunji Chen and Olivier Temam：*DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning*, Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Association for Computing Machinery(2014).

[3] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao and Jason Cong：*Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks*, booktitle = Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Association for Computing Machinery(2015).

[4] Jorge Albericio and Patrick Judd and Tayler Hetherington and Tor Aamodt and Natalie Enright Jerger and Andreas Moshovos：*Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing*, Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA), IEEE(2016)

[5] Karu Sankaralingam：*Mozart: Designing for Software Maturity and the Next Paradigm for Chip Architectures*, The 33rd Symposium of Hot Chips (HC), 2021

[6] Raghu Prabhakar and Sumti Jairath：*SambaNova SN10 RDU: Accelerating Software 2.0 with Dataflow*, The 33rd Symposium of Hot Chips (HC), 2021

[7] Reiner W. Hartenstein, Rainer Kress and Helmut Reinig：*A New FPGA Architecture for Word-Oriented Datapaths*, Proceedings of the 4th International Workshop on Field-Programmable Logic and Applications: Field-Programmable Logic, Architectures, Synthesis and Applications, Springer-Verlag(1994).

[8] R. Hartenstein, R. Kress and H. Reinig：*A reconfigurable data-driven ALU for Xputers*, Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines (FCCM), IEEE(1994).

[9] Ray Bittner and Peter Athanas：*Wormhole Run-Time Reconfiguration*, Proceedings of Fifth International Symposium on Field-Programmable Gate Arrays (FCCM), Association for Computing Machinery(1997)

[10] S.C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R.R. Taylor and R. Laufer：*PipeRench: a co-processor for streaming multimedia acceleration*, Proceedings of the 26th International Symposium on Computer Architecture (ISCA), IEEE(1999).

[11] André DeHon, Yury Markovsky, Eylon Caspi, Michael Chu, Randy Huang, Stylianos Perissakis, Laura Pozzi, Joseph Yeh and John Wawrzynek：*Stream computations organized for reconfigurable execution.*, Microprocess. Microsystems 30(6), Elsevier(2006).

[12] Y. -H. Chen, J. Emer and V. Sze：*Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks*, Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA), IEEE(2016).

[13] H. Kung：*Why Systolic Architectures?*, Computer, vol.15, no.01, IEEE(1982).

[14] N. Jouppi, et al.：*In-datacenter performance analysis of a tensor processing unit*, Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), IEEE(2017).

[15] V. Govindaraju, C. Ho and K. Sankaralingam：*Dynamically Specialized Datapaths for energy efficient computing*, IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), IEEE(2011).

[16] Tony Nowatzki, Newsha Ardalani, Karthikeyan Sankaralingam and Jian Weng：*Hybrid Optimization/Heuristic Instruction Scheduling for Programmable Accelerator Codesign*, Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (PACT), No.15, Association for Computing Machinery(2018).

[17] Raghu Prabhakar, Yaqi Zhangand, David Koeplingerand, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedramand, Christos Kozyrakis and Kunle Olukotun：*Plasticine: A Reconfigurable Architecture For Parallel Paterns*, Proceedings of the 44th Annual International Symposium on Computer Architecture(ISCA), No.14, Association for Computing Machinery(2017).