情報処理学会論文誌　コンピューティングシステム　Vol.5　No.3　74–85 (May 2012)

画像処理向け線形アレイアクセラレータの性能評価

中田 尚[1]　吉村 和浩[1]　下岡 俊介[1]　大上 俊[1]
デビセッティ ベンカタ ラマ ナヴィーン[1]　中島 康彦[1,a)]

受付日 2011年10月7日, 採録日 2012年1月21日

概要：近年，高性能組込み機器のライフサイクル短縮と開発コストの増加にともない，専用ハードウェアを採用することが難しくなっている．一方，ソフトウェアにより機能を柔軟に変更できるプロセッサの採用を考えた場合，最低性能保証のために余裕を持ったコア数と並列処理プログラムの投入，低電力化のための一部専用ハードウェア化などの工夫が必要であり，解決すべき課題は多い．我々は，このような要求に対する 1 つの解決策として，安定的な高性能とソフトウェア互換性の両立を目指す線形アレイアクセラレータを提案している．本アクセラレータは，一般的な VLIW プロセッサを線形アレイ型に拡張したうえで，最内ループをアレイ構造に写像し，ループ内のすべての命令を演算器ネットワークに固定的にマップし，入力データを順次流し込むことにより，ループの複雑さによらず毎サイクル出力を得ることを特長としている．本稿では提案アクセラレータ全体の詳細構成と性能評価について述べる．RTL シミュレータを用いた評価の結果，5 種類の主要な画像フィルタの実行には 30 段構成で十分であり，現実的なメモリアクセスレイテンシを仮定した場合においても，既存の VLIW プロセッサと比較して最大 16.2 倍の性能を達成可能であることが分かった．

キーワード：アクセラレータ，画像処理，VLIW

A Linear Array Accelerator for Image Processing

Takashi Nakada[1]　Kazuhiro Yoshimura[1]　Shunsuke Shitaoka[1]　Suguru Ooue[1]
Devisetti Venkata Rama Naveen[1]　Yasuhiko Nakashima[1,a)]

Received: October 7, 2011, Accepted: January 21, 2012

Abstract: Recently, the requirements for low power and stable high performance are increased rapidly. Moreover, developing special-purpose hardware is not acceptable due to the short life cycle and enormous manufacturing cost. Therefore, processors that can achieve stable high performance and execute existing programs are desired. We have proposed a Linear Array accelerator that concatenates several VLIW processors in a linear fashion. All instructions of most inner loop are mapped to the processor network, and the input data is supplied continuously, so that the output data is produced every clock cycle. In this paper, we designed the accelerator and estimated it's performance using an RTL simulator. For five major graphic filters, 30 stages structure is required. The result shows the processor can achieve 16.2-fold speedup as compared to an existing VLIW processor.

Keywords: accelerator, image processing, VLIW

1. はじめに

近年，組込み機器が取り扱う情報量が増大し，低消費電力かつ最低性能保証可能なプロセッサの需要が急速に高まっている．これまで，最低性能の保証は，専用ハードウェア化による実現が一般的であった．専用ハードウェアでは，必要となる機能モジュールを組み合わせることにより，要求性能を満たしている．ここで最も重要なことは，定常的に結果を出力できることであり，個々のモジュールに対し

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[a)] nakashim@is.naist.jp

て，過剰な性能は不要である．つまり，タスクの複雑さによらず一定のスループットが得られるようにすることで要求性能を実現しており，その結果，各モジュールのハードウェア量はタスクの複雑さに比例する．

しかし最近では，次々に策定される画像や無線などの新規格に追随したり，製品差別化のためのフィルタ微調整や出荷後の機能更新に対応したりするために，時間的・経済的コストが許容できない応用分野においては，専用ハードウェア以外の選択肢が模索されている．すなわち，専用ハードウェアに匹敵する性能を有し，かつ，ソフトウェアにより機能を柔軟に変更できる高性能かつ柔軟なアーキテクチャが求められている．

このような状況を打破する手段として現在有望視されているのが，**マルチコア**や**メニイコア** [1], [2], [3]，および，**リコンフィギャラブルデータパス** [4] である．前者は，様々な粒度に分割したアプリケーションプログラムを複数コアにより並列実行するアーキテクチャであり，一般的な並列プログラミング手法が使えるという利点がある．また，ソフトウェアが制御可能な局所メモリを備えることにより，予測不能なキャッシュミスによる性能低下をある程度抑制できるため，柔軟性と拡張性に優れるだけでなく，コア数増加による最低性能保証も視野に入れることが可能である．一方後者は，プロセッサコアよりも粒度の小さい演算器を多数配置し，**機能の柔軟性**と**高速性**の両立を図るアーキテクチャである．膨大な演算器を配置することにより，専用ハードウェアと同様に，比較的低い動作周波数でも高性能プロセッサに匹敵する性能を達成しうるという利点がある．

ところで，両者の得失を比較すると，対極の関係にあることが分かる．前者には，一般的な並列プログラミング手法が使える柔軟性がある代わりに，開発時に正確に予測することが困難なバス競合などの性能低下分を補償するため，余裕を持ったコア数の投入が必要である．後者には，命令フェッチ機構を省略でき，要求性能と演算量から必要なハードウェア量を決定できるという利点がある代わりに，機械語命令を実行する汎用プロセッサと大きく異なる構成をとっており，既存の機械語命令で記述された資産（開発環境を含む）を活用することが困難である．もし，既存プロセッサと互換性のある機械語命令を用いて，膨大な演算器を効率良く動作させ，専用ハードウェアのようにタスクの複雑さに比例したハードウェアを投入することにより，**安定したスループットを実現**できる仕組みを構築できれば，冒頭に述べた課題に対する直接の解となりうる．

以上の考えに基づき我々は**線形アレイアクセラレータ**を提案している．文献 [5] では提案アクセラレータの核となる演算器アレイへの命令割当て手法とそれに基づく演算器ネットワークの実現可能性について検討を行い，既存プロセッサと同等の動作周波数を実現可能であるとの結論を得た．本稿ではさらに詳細な実装を行い，制約条件を明らかにするとともに，提案アクセラレータ全体の回路規模と画像処理ベンチマークを用いた性能評価を行う．

以降，2 章で関連研究について述べ，3 章では，線形アレイアクセラレータの概要および既存命令列をアレイ動作により高速実行するモデルについて述べる．4 章では動作の詳細について述べ，5 章では画像処理を用いた定量的評価を行う．

2. 関連研究

ソフトウェアとの親和性，および，リコンフィギャラブルデータパス [4] による高速性の両立を目指す仕組みに，粗粒度リコンフィギャラブルアレイ（CGRA）がある．ソフトウェアとの親和性の観点から，低電力 ILP プロセッサの代表格である **VLIW** [6] が用いられることが多く，VLIW 資産やコンパイラ技術の活用による機械語命令との連続性が期待される．VLIW を用いることにより，既存技術の延長でさらなる並列性の抽出を目指す点は多くの関連研究で共通しているが，その実現方法は様々である．

たとえば，ADRES [7], [8] では 4 命令発行の一般的な VLIW 構成のバックエンドとして，各演算ユニットにローカルレジスタファイルを備えた 4×3 程度の CGRA を接続し，専用コンパイラが，制限付きの C 言語により記述されたプログラムから，4×3 命令発行に対応する CGRA 命令を生成する．CGRA は VLIW の関数 CALL 命令により起動される．すなわち VLIW 命令に対する上位互換性を備えている．

PPA [9] では，2×2 の CGRA を 8 コア接続し，ループカーネルに応じてアレイサイズの構成を変更できる．そして，異なるループカーネルをパイプライン実行することで，対応可能なループサイズの拡大と演算器の動作率の向上の両立を実現している．

PipeRench [10]，PARS [11]，LSRDP [12] や FE-GA [13] では，より大規模な CGRA に対して DAG を用いたスケジューリングを適用することでさらなる性能向上を目指しているが，大規模であるがゆえに既存プロセッサとは大きく異なる構成となっており，たとえば全体の回路規模増大を抑えるために，演算器アレイ中に配置されるレジスタ数が限定され，**高度なスケジューリング機能**が要求される．

また，これらの CGRA にはスケーラビリティに問題がある．より複雑なタスクをスループットを維持したまま実行するためには，さらに多くの演算器を投入しなければならない．しかし，演算器数の増加により演算器間ネットワークが複雑化するとともに，専用コンパイラによるプログラム生成やスケジューリングの難易度も高くなることが容易に予想されるが，このような問題にどこまで対応できるかは明らかではない．

一方，本稿では，CGRA 命令のような専用命令を使用

せず，既存コンパイラが生成する命令列を利用する線形アレイアクセラレータを提案する [14]．これにより既存コンパイラを流用することができる．そして，後述するようないくつかの制約はあるものの，制約を満たしているアセンブリ命令列であればそのままアレイ演算器で実行されるため，プログラム開発者にとっても専用言語の習得や特別なプログラミングの知識は不要であり，プログラム開発自体に注力することができる．また，提案アクセラレータは既存プロセッサを線形に拡張することにより，演算器間ネットワークの複雑化を抑え，スケーラビリティの問題を解決する．アレイ構造の起動には，既存のキャッシュプリフェッチ命令のみを使用しており，上位互換性[*1]だけでなく，アレイ構造で動作するロードモジュールを従来のプロセッサ上でも実行できる下位互換性[*2]も備えている．このような互換性を利用することにより，従来プロセッサで動作しているプログラムの中から必要な箇所のみを順次アレイ実行に対応させるといった段階的なプログラム開発も可能となる．また，演算器ネットワークを頻繁に切り替える既存 CGRA と異なり，アレイ動作中は演算器ネットワークを固定することにより，命令キャッシュを含むフロントエンド部分を完全に停止し，演算器アレイのみで通常動作時と等価な演算を高速実行する．

拡張の基となるベースプロセッサには一般のスーパスカラプロセッサを用いることができる．演算器アレイ構造では 2 次元構造を持ち，命令発行幅がアレイ構造の幅を決定するため，効率の良い実装のためにはある程度広い発行幅が望ましい．しかし，発行幅の広いスーパスカラプロセッサには大規模な命令スケジューリング機構が不可欠であり，動作周波数や回路規模の点で不利である．そこで我々はベースプロセッサとして VLIW プロセッサを採用する．これにより大規模な命令スケジューリング機構が不要になるとともに，コンパイラによってスケジューリング済みの命令列をそのままアレイ構造にマップするため，演算器アレイへのマップ結果を容易に推測できる．これにより，たとえば，コンパイラの出力した機械語コードを見るだけでアレイ実行に対応可能かどうかを判断できる．

3. 線形アレイアクセラレータ

ここでは，実際のプログラム例を用いて，提案アクセラレータの概要とその実行モデルについて述べる．

3.1 概要

図 1 に，3 × 3 の画素から輪郭抽出を行う VLIW 命令列の一例を示す．各画素は RGB 各 1 バイトを含む 4 バイトとし，対角画素のバイトごとの SAD（Sum of Absolute Difference）の総和が閾値を超えた場合，中央座標に輪郭があると判断している．命令語中の i は入力画素中央の主記憶アドレス，o は出力画素のアドレス，W は画面の幅，k，1 から 8 と，x，y，z，w，p，s は各々レジスタを表現している．2 番目の分岐命令はループカウンタ（k）が 0 になった際のループ脱出用，最終命令の分岐命令はループ先頭への復帰用である．CMP 命令は SAD 総和（p）と閾値（T）を比較して条件コード（c）をセットし，SEL 命令は条件コードに従い輪郭情報（s）に 0 または 255 をセットする．この命令列を理想的に実行した場合，10 サイクルごとに 1 画素の輪郭情報を生成できる．さらに，このプログラムにループアンローリングを適用すると，参照される画素には再利用性があるため，1 ループあたりの LD 命令の数は 3 個となる．その結果，SAD 命令が実行速度を律速し，4 サイクルごとに 1 画素の輪郭情報を出力できる．

提案アクセラレータの特徴は，従来型 VLIW では演算器バイパスとして実現される演算器間ネットワークを複数段の演算器アレイに展開し，命令デコーダを流用してループ構造の全命令を各演算器に写像したうえで，命令デコーダを含むフロントエンドを停止した点にある．ループの回転数に相当する入力データをキャッシュから初段へ流し込むことにより，既存プロセッサと互換性のある機械語命令を用いて毎サイクル 1 画素の出力データ（輪郭情報）を得られる点にあり，この特性を利用することにより性能予測が容易になる．このとき，安定したデータ供給を保証するため，ループ中でアクセスするデータをあらかじめ L1 キャッシュに正確にプリフェッチする必要がある．

ただし，任意の VLIW 命令列を演算器アレイに写像するには，任意の段にロード命令を配置できる必要がある．しかし，数多くのロード命令が L1 キャッシュを直接参照することは，キャッシュメモリに必要なポート数の点から非現実的である．このため，論理的には全段が共通の L1 キャッシュの内容を参照できるものの，物理的には各ロード命令は同一段の小容量 L0 キャッシュのみを参照する巧妙な仕組みが必要である．同様に，各段が必要とするデータがすべて前段の演算器からのバイパスにより供給される理想的な命令列を前提とすることはできないため，各段の演算器がレジスタを直接参照できる仕組みが必要である．しかし，多くの演算器が 1 組のレジスタファイルを参照することは，L1 キャッシュと同様に非現実的である．このため，レジスタファイルについても，論理的には同一レジスタ空間を参照できるものの，物理的には各段に分散配置する仕組みが必要となる．

3.2 構成

提案アイデアに対して以上の物理構造を反映させた結果，**図 2** に示すように，一般的な VLIW プロセッサを初段

[*1] すでに既存のプリフェッチ命令が挿入されていたとしてもアレイ実行には対応不可能であるばかりか，誤動作の可能性があるので適切な変換を行うか取り除くかしなければならない．
[*2] プリフェッチ命令の解釈の違いにより，想定される性能を発揮することができない可能性がある．

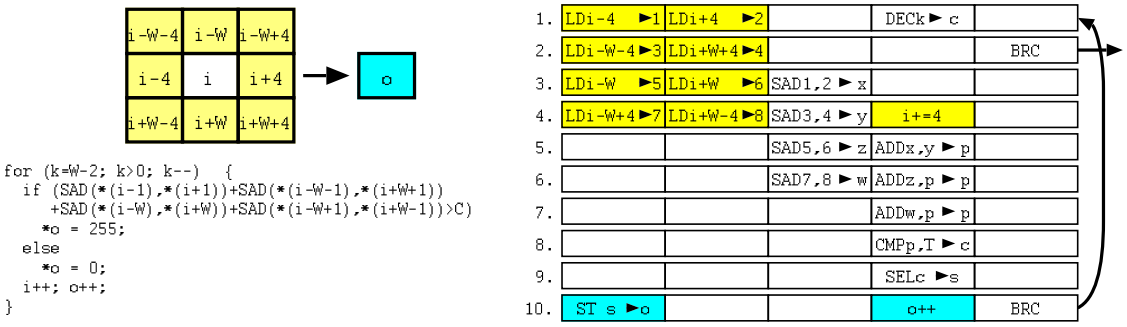

図 1　VLIW による輪郭抽出

Fig. 1　Edge detection written in VLIW.

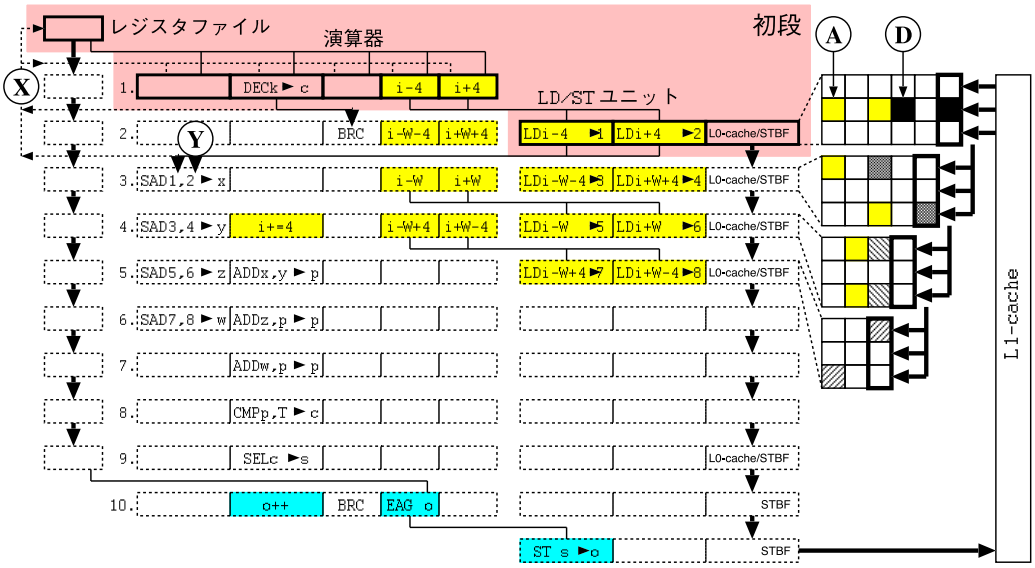

図 2　線形アレイアクセラレータによる輪郭抽出

Fig. 2　Edge detection with a linear array accelerator.

に配置し，次段以降にレジスタファイル，演算器，ロードストアユニット，小容量キャッシュおよびストアバッファの組を配置した**アレイ構成**に至る．ここで，LD/ST 命令はアドレス計算と実際のデータ転送の 2 サイクルに分けて実行されるとした．L1 キャッシュは，初段および最終段とのみ接続されており，段数に関するスケーラビリティが高く，要求に応じて自由に段数を変化させることができる．具体的には，初段と最終段が L1 キャッシュに隣接するようなリング構造を想定する．万一，十分近い距離に配置できない場合には，初段からの読み書きを優先し，最終段からの書き込みパスを 1 サイクル遅らせればよい．スループットが確保されていれば，この書き込みの遅延はアレイ実行 1 回につき 1 度だけ発生するペナルティとなるので，十分に許容できる．また，前述の ADRES [7], [8] が初段のみにキャッシュを備え，演算結果を最終的に初段に集約する必要があるのに対し，本提案ではデータが 1 方向に流れるため，段数によらず隣接する段間の接続は一定であり，演算器ネットワークの複雑化を回避している．具体的には，従来型 VLIW におけるバイパスネットワークを自分自身ではなく次段の演算器群へ接続する．各演算器では自分自身へのフィードバックループは依然として必要であるが，1 対 1 の接続であるので，バイパスネットワークと比較して十分小さな増加量である．図 2 では，図 1 に示した VLIW 命令の各々が，各演算器に対応付けられている．初段のレジスタファイルから読み出した値を用いて，初段の演算器が，VLIW 命令（1.）の DEC 命令と，LD 命令の実行に必要なアドレス計算（`i-4` および `i+4`）を行う．アドレス計算結果に基づき，初段の LD/ST ユニットが内蔵する L0 キャッシュを参照する．通常の VLIW 動作では，ロード結果が初段のレジスタファイルに書き込まれ，必要に応じて初段の演算器にバイパスされる（X）．一方，アレイ動作では，ロード結果が VLIW 命令（3.）の SAD 命令に対応付けられた演算器にバイパスされる（Y）．

アレイ構造の 1 段を通過するのに 1 サイクルを要する場合，VLIW 命令（1.）において最初のロードを開始してから，VLIW 命令（10.）において結果がストアされるまでに 10 サイクルを要する．この点では，従来の VLIW プロセッサによる実行と何ら変わらない．しかし，全段をパイプライン動作させることにより，各演算器に異なるイタレーションの同一命令を毎サイクル実行させ，最終的に，

VLIW 命令（10.）のストア命令を毎サイクル実行できる（以降ではこの状態を**一斉演算**と呼ぶ）．本提案の特長は，最終的な演算結果生成のスループットが，VLIW 命令列の長さに依存しない点にある．十分な長さのイタレーション回数があれば，命令列が長いことによる立ち上がりオーバヘッドは隠蔽される．一方，アレイ構造の段数により，収容可能な VLIW 命令列の長さが制限される点は，従来のプロセッサにない制約である．一般的な画像処理のために，どの程度の段数が必要であるかについては，5.1 節で評価する．

さて，レジスタや L0 キャッシュが分散していても，同一イタレーションに関与する VLIW 命令に対して，いずれの段においても同一内容を見せることができれば，通常 VLIW 動作と論理的に同じ演算結果が得られる．たとえば初段の VLIW 命令（1.）の LD 命令（`i-4`）が，第 1 イタレーションに対応して L0 キャッシュ中の A を参照したとする．3 サイクル後に初段の i が 12 増加し，この命令は第 4 イタレーションに対応して D を参照する．このとき，第 4 段の VLIW 命令（4.）の LD 命令（`i+W-4`）では，初段の VLIW 命令（1.）が最初に用いた i を使用して，第 1 イタレーションの最後の LD 命令を実行する．このためには，アレイ構造に配置した VLIW 命令間を演算結果が伝搬するのと同じ速度で，レジスタ，L0 キャッシュおよびストアバッファの内容を併走させればよい．具体的には，ある瞬間における隣り合った段の L0 キャッシュが保持すべきデータの差分のみを初段に供給し，それを後段に順次転送していけばよい．この例では毎サイクル 3 画素分のデータが段間を転送されるため，L1 キャッシュからも毎サイクル 3 画素分のデータを読み出す必要がある．また，初段の L0 キャッシュに十分データが蓄えられるまでの間（この例では 3 サイクル）は演算開始を待機する必要がある．以上のように，A を含む 3×3 の領域を初段から第 4 段に順に伝搬させることにより，初段から第 4 段までの LD 命令は，順に，物理的空間は異なるものの内容が同じ各々の L0 キャッシュを参照して，第 1 イタレーションに対応した 8 個の LD 命令を正しく実行できる．

また，LD 命令のアクセス先が L1 キャッシュではなく，より容量の小さい L0 キャッシュになるため，LD 命令 1 回あたりの消費電力の削減も期待できる．その一方で，消費電力が増加する追加回路もいくつか存在する．まず，L0 キャッシュのデータを正しく保持するためには，初段から LD 命令が存在するまでの間のデータ転送が必要である．また，演算器アレイでは元の命令列で離れた命令は物理的にも離れた演算器に配置される．通常プロセッサでは，離れた後続命令で使われるデータはレジスタファイルに保持されるが，提案アクセラレータではデータはアレイ中を通ってデータが伝搬される．このような追加機構のために必要となる電力については詳細な評価が必要であるが，現在の想定ではその回路規模からそれほど大きなものにはならないと考えている．また仮に，消費電力が増加したとしても性能の向上幅がそれよりも十分大きければ，電力効率は向上する．

3.3　制約条件

前述のアレイ段数以外にも提案アクセラレータでプログラムを高速実行するためには，いくつかの制約条件がある．ここでは，これらの詳細について述べる．

アレイ段数

ループ中の VLIW 命令数がアレイ段数よりも多いときは，そのままでは演算器アレイで実行できないため，事前にループを分割するなどの対応が必要である．もし，実行時にアレイ段数を超えていることが判明した場合は，演算器アレイでの実行を中止し初段のみによる通常実行を行う．一般的な CGRA においても，構成情報を保存するメモリ容量などにより実行可能な命令数には制限があるものが多い．

依存関係

i 回目のイタレーションの実行結果を i + 1 回目のイタレーションで参照する場合のような，イタレーションを超える依存関係が存在する場合は，基本的に演算器アレイで実行することはできない．これは配線の複雑化のような物理的な問題だけではなく，そもそも複数のループイタレーションが並行に実行されているため，結果が必要となる時刻においてその結果を出力する演算が実行されていない可能性があるためである．これは多数の演算器を用いる場合にはつねに発生しうる問題であり，根本的な解決のためにはアルゴリズムレベルでの変更が必要となる．

ただし，このような依存関係をまったく考慮しない場合にはループカウンタの更新すらできなくなってしまうため特別な対応が必要である．具体的には `i+=N` で表現される命令（以降，自己更新命令）の実行をサポートする．このためには，演算器の出力を自分自身の入力にフィードバックする回路の追加を行えばよい．一斉演算時には初回のみ前段から初期値 `i` を受け取り `N` を加える．次のサイクルからは追加したフィードバック回路を通して自演算器の出力を受け取り，それに `N` を加えればよい．この構造から `N` はループ中で変更されない定数であることも必須条件である．

ループの形態

前述のように演算器アレイにデータを正確にプリフェッチするためには，ループの実行回数が既知である必要がある．もし，プリフェッチサイズが L1 キャッシュの容量を超過する場合には，ループを複数回に分割する必要があるが，これは既存プロセッサにおけるブロック化のようなキャッシュ最適化と同等の難易度である．さらに，簡単な拡張でループ途中での脱出をサポートすることは可能であるが，その場合もループ回数の最大値を保証する必要があ

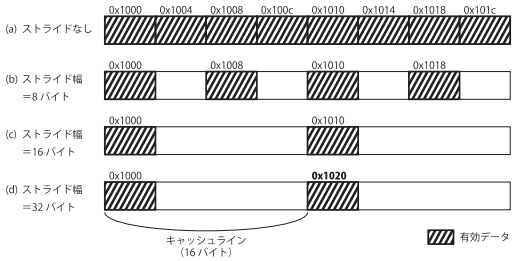

図 3　ストライド付きプリフェッチ
Fig. 3　Prefetch for stride access.

る．また，この場合はプリフェッチデータが無駄になる可能性があるので，そのオーバヘッドを慎重に評価する必要がある．本稿では定数回のループのみを評価対象とする．

ロード命令

ロード命令に対するデータ供給は，前述のように分散して配置した L0 キャッシュによって行う．そのため，ループ中で実行可能なロード命令にはいくつかの制約がある．まず，データを正しくプリフェッチするためには，すべてのロード命令のアクセスパターンが解析可能である必要がある．具体的には連続アクセスか，2 の冪乗の定数幅のストライドアクセスの場合のみをサポートする．また容量の制約から，1 つのイタレーション中でアクセスされるデータが各 L0 キャッシュに収まる必要がある．特にストライドアクセスが要求される場合には，通常のキャッシュでは有効容量が「要素サイズ」/「ストライド幅（バイト）」に減少してしまう．そこで，できる限り多くの要素を格納できるように自動的に不要な領域を詰めて L1 または L0 キャッシュに格納する構成とした．**図 3** は要素サイズが 4 バイト，ラインサイズが 16 バイトの場合の例である．しかし通常のキャッシュと同様に，同一のキャッシュライン中には連続する領域のデータのみが格納可能であるため，ストライド長がキャッシュライン幅を超える場合にのみこの機構は有効となり，L1/L0 キャッシュの有効容量の減少を「要素サイズ」/「キャッシュライン幅」に抑えることができる (図 3 (d))．もし，これらの影響でプリフェッチが不完全になる場合にはループを分割する必要がある．

ストア命令

ストア命令に関してもロード命令とほぼ同等の制約条件がある．特にストアバッファの制約から，1 イタレーションあたり 1 ワードまでのストアが可能である．後述のように，合計が 1 ワードに収まっていれば，バイト単位のストアも可能である．データ入出力の詳細については 4.2 節で述べる．

中間データの参照

実装を単純にするために，ループ中でレジスタに保持される中間データがループ終了後に参照されないという制約を設けている．この制約の詳細については 4.1 節で詳しく述べる．

以上の制約条件をすべて満たしたループのみが演算器アレイによる高速実行の対象となる．

4. アレイ動作の詳細

本章では，アレイ動作について動作の概要を述べた後に，データ入出力機構の詳細について説明する．

4.1 動作の概要

アレイ動作は，BASE，DIST1 および DIST2 をオペランドとするプリフェッチ命令により開始され，演算器・レジスタネットワーク設定を行う**アレイ設定**，L1 プリフェッチと Dirty 追い出しを行う**プリフェッチ**，一斉演算を行う**アレイ実行**を経て通常 VLIW 動作へ復帰する．このうち，アレイ設定とプリフェッチは同時動作が可能であり，両方が完了し L1 キャッシュに対するロードおよびストアに関してキャッシュミスが発生しないことが保証できた段階でアレイ実行に移行できる．アレイ設定では，演算器アレイ段数分のサイクル数を使って，デコーダ結果を各段の演算器・レジスタネットワーク設定 FF（フリップフロップ）に順にセットする．プリフェッチでは，「新規に必要なワード数」/「メモリバスのバンド幅」分のサイクル数を使って，プリフェッチ命令のオペランドに基づき外部メモリから L1 キャッシュへの転送を行う．L1 キャッシュから外部メモリへの Dirty ライン追い出しについても同様に「追い出しが必要なワード数」/「メモリバスのバンド幅」分のサイクルで追い出しを行う．たかだか数十段のアレイ構造に比べてアレイ動作が 1 度に処理するワード数が多い場合，アレイ設定のオーバヘッドはプリフェッチの実行時間によって隠蔽される．

まずアレイ設定では，プリフェッチ命令の次命令から無条件後方分岐を検出するまでの全命令を演算器アレイにマップする．命令マッピングの詳細については文献 [5] を参照されたい．アレイ設定中にループ中の命令数がアレイ段数よりも大きいことが判明した場合には，アレイ実行に移行することなく通常実行に戻る．アレイ設定では演算は開始されていないため，すべてのアーキテクチャ状態はループ開始前の状態を保持しており，すぐさま通常実行に戻ることができる．

ループ脱出条件が成立し，一斉演算停止指示が最終段に到達した時点で最終イタレーションの実行が完了するので，最終段が初段に対して一斉演算完了を通知し，初段は，通常 VLIW 動作に復帰する．このように，デコーダ，初段および最終段のみを直接接続してアレイ動作を制御することにより，段数に関するスケーラビリティを維持できる．

次にプリフェッチでは，各段におけるロードに際してキャッシュミスが発生しないことを保証するプリフェッチ，および，最終段におけるストアに際して同様の保証をする Dirty 追い出しを行う．これは，一斉演算中のキャッシュ

ミスに対応するには，一斉演算の一時停止/再開機構が必要となり，制御がきわめて複雑化すると考えられるためである．一斉演算中に必要となるデータをすべてL1キャッシュにプリフェッチし，一斉演算中にはL0キャッシュを介してデータを順次供給することにより，キャッシュミスが起こらないことを保証し複雑な機構を省略できる．本稿では，3 × 3程度の画素に対して画像処理を繰り返すことを想定しており，縦3画素（前述のDIST2に対応）× 水平方向総画素数（同DIST1に対応）分の入力画素値をL1キャッシュにプリフェッチでき，かつ，縦1画素 × 水平方向総画素数分の演算結果をL1キャッシュにストアできればよい．水平方向に1行分の処理が終了するごとに，処理対象が垂直方向に1画素分ずつ移動することから，3行分の入力画素値のうち2行分は再利用でき，新たな1行分のみをプリフェッチすればよい．3行分の入力画素値と1行分のストア先を常時確保するために，キャッシュを4つのブロックに分割し，新たな1行分をプリフェッチする際には，ブロックをサイクリックに使用し，ストアに使用したブロックに対して次のプリフェッチを行うことにより，キャッシュを効率的に利用できると考えた．

そしてアレイ実行では，L1キャッシュから初段のL0キャッシュへのデータ転送開始を契機として，初段から順に演算開始信号を伝搬させ，一斉演算を開始する．3 × 3の画素（各画素1ワード）を入力とし，毎サイクル水平方向に1画素ずつ移動しながら1画素の結果を出力する場合，毎サイクル3ワードのデータ転送が必要である．L1キャッシュと同様にL0キャッシュも4つのブロックに分割し，同一番号のブロック間においてデータ転送を行うことにより，キャッシュ間バス構成の複雑さを低減させることができる．一斉演算中は，ループイタレーションの先頭に対応するVLIW命令を初段の演算器において実行する．この際，初段のレジスタファイルおよびL0キャッシュの内容を参照し，ストア命令は同一段のストアバッファに格納する．次のサイクルでは，次のVLIW命令を次段の演算器において実行する．この際，次段のレジスタファイル，L0キャッシュおよびストアバッファの内容を参照するものの，従来型プロセッサの演算器バイパスと同様に，初段の実行結果も参照できるとする．すなわち，前述のように，論理的に，初段のレジスタファイル，L0キャッシュおよびストアバッファのすべての内容が1サイクルで次段にコピーできる必要がある．実現のためにどのような工夫が可能であるかについては次節において述べる．次段の演算と同時に，初段では，次のイタレーションの先頭に対応するVLIW命令を実行する．

最後に，ループを脱出する条件分岐命令の成立を検出すると，一斉演算の終了処理を開始する．このとき，分岐命令よりも後段では前のイタレーションを実行中であるため，すぐに全体を停止するのではなく，演算終了信号を順次後段に伝えることにより，終了処理を行う．最終イタレーションの結果が書き込まれるタイミングで，終了信号が最終段に到達しアレイ実行が終了する．

このように，一斉演算中は，各段が異なるイタレーションの演算を同時に行う．また，各段は毎サイクル同一命令を実行するために，命令デコードの必要がなく，分岐予測や命令フェッチに関するフロントエンド部分を完全に停止することができる．

また，現在の実装ではアレイ実行中にストア命令で更新されるメモリ上の値は保証するが，レジスタの値は保証しない．これは，レジスタに保存されている値は計算の途中結果が主であり，以降の実行で参照されないからである[*3]．最終段から初段のレジスタへ書き込みを行うための回路を追加し，アレイ実行の終了処理中に書き戻しを行えば，アレイ実行終了時の値を保証することも可能であると考えるが，現在の設計・実装では省略している．

4.2　データ入出力機構

これまでに述べたように，本稿が提案する実行モデルでは，各段のレジスタ，L0キャッシュ，および，ストアバッファの全内容を毎サイクル次段へコピーする必要がある．しかし，このような機構を額面どおりに実現することきわめては困難である．以降では，実装可能な手段を用いて論理的に同等の機能を実現する方法について述べる．

3.2節で述べたように，初段のL0キャッシュのうち毎サイクル更新されるデータは，L1キャッシュから送り込まれるデータのみである．このため，L0キャッシュの全内容を次段へコピーする必要はなく，L1キャッシュから送り込まれた内容のみを次段へ送れば十分である．すなわち，各イタレーションが3 × 3の画素を扱う場合，毎サイクル3ワードのみを次段のL0キャッシュに転送すればよいため，実現は容易である．2段目以降についても同様に，前段から送り込まれたデータのみを次段へ送ればよいため，段数の増加に対しても柔軟に対応可能である．さらに，個々のワードについてはそれぞれがウェイに固定的に対応するため，段間での転送やL0キャッシュへの取り込みにおいて1対1の単純な接続でよい．

また，任意段においてストア命令を実行可能とするために，各段にストアバッファを設ける必要があるものの，一般的な画像処理を考えた場合，L1キャッシュに対するプリフェッチ性能以上にストア性能を確保する必要はない．すなわち，任意の段が生成したストアデータを各段が自由にL1キャッシュへ書き戻せる必要はなく，次段へ順次伝搬させ，最終段のストアバッファからL1キャッシュに毎サイクル1画素を書き出せればよい．このためには，各段につき1ワードのストアバッファがあれば十分であり，次

[*3] 今回用いた評価プログラムにおいてもこれらの値が参照されることはない．

段への全コピーは容易に実現できる．なお，この構成は，ストア命令の記述を 1 カ所に制限するものではない．前段から伝搬されるストアバッファに対し，異なるワードアドレスへのストアについては全体を上書きすることにより，前段のストアデータを後段においてロードするようなスピルアウト/インや，汎用レジスタとメディアレジスタ間のデータ転送に対応できる．また，同一ワードアドレスへの部分ストアについては，順にマージすることにより，バイト単位に生成した演算結果の逐次ストアに対応できる．すなわち，部分ストアを開始する前であれば，1 ワードのストアバッファを利用して任意回のストア→ロードを実行できる．

図 4 に，以上の検討の結果得られる構成を示す．4 つの way に分割された右端の L1 キャッシュのうち，3 つの way から初段の L1 読み出しバッファに対して，毎サイクル 3 ワードを読み出す．次のサイクルにおいて，初段の L0 キャッシュと次段の L1 読み出しバッファに転送し，以下同様の手順により，後段の L0 キャッシュと L1 読み出しバッファに順次転送する．ストアバッファについても，新たなストアデータを反映しながら，後段に順次転送する．

以上の構成から，提案アクセラレータでは，ループアンローリングのように既存のプロセッサに対して非常に有効であった最適化手法が事前に適用されていると，1 イタレーション中の LD/ST 回数が増えすぎてしまい，アレイ実行ができない．この問題を回避するためには，単にループアンローリングの適用を行わないようにすればよい．

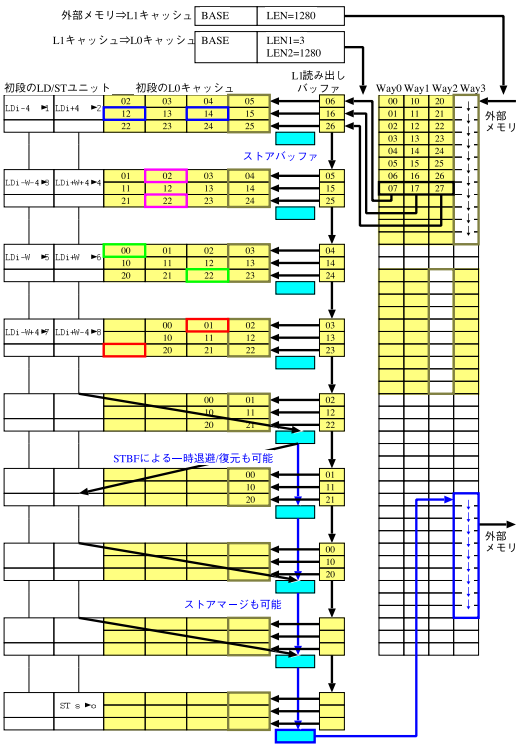

図 4　L0 キャッシュとストアバッファの伝搬
Fig. 4　Propagation of L0-caches and store buffers.

5.　画像処理を用いた評価

本章では，画像処理プログラムおよび RTL シミュレータを用いて，線形アレイ VLIW アクセラレータの評価を行う．プログラムには，画像拡大（Z），鮮鋭化（S），メディアンフィルタ（M），輪郭抽出（E），輪郭ノイズ除去（N）を用いた．画像拡大では，近傍 4 画素から線形補完で目的画素を得る．鮮鋭化では近傍 9 画素にアンシャープフィルタを適用する．メディアンフィルタでは近傍 9 画素の中央値を求める．輪郭抽出では輪郭強調フィルタを適応した後に閾値を用いて 2 値化する．輪郭ノイズ除去では近隣 9 要素の平均値が閾値未満の場合に該当要素をノイズと見なし 0 とする．このうち，S，M，E は図 2 で示した例と同様に 9 画素の入力から 1 画素の出力を得るが，Z では入出力画像のサイズが異なり，N では画素の代わりに 1 バイトの輪郭情報を入出力とする．そのため，N では 1 イタレーションで 4 要素を出力する．各画素は RGB 各 1 バイトを含む 4 バイト（N を除く），画像サイズは 320 × 240 である．各プログラムについて，C 言語により記述し FR-550 用の gcc-4.1.2，最適化オプション ‘-O3’ によりコンパイルしたものを用いたが，そのままでは命令密度が低く不当に提案手法に有利なため，MEDIA 命令を用いてループカーネルを書き直し，(1) ループアンローリングを適用したもの，(2) アレイ動作のために最内ループの直前にプリフェッチ命令を挿入[*4]したものを用意した．ここで MEDIA 命令は既存プロセッサを参考にした独自仕様であり，SAD（Sum of Absolute Difference）命令や最大値，最小値を求める演算，バイト単位の飽和演算のような SIMD 命令が含まれる．RTL シミュレータは，4 章で述べた動作モデルに従い，プリフェッチ命令を検出して通常 VLIW 動作からアレイ動作へ移行する．すなわち，(1) については通常 VLIW 動作により実行し，(2) についてはアレイ動作を併用して実行する．

表 1 に提案アクセラレータの諸元を示す．なお，これまで，図を見やすくするために，各段に LD/ST ユニットを 2 個装備して段数を減らしていた．しかし，各段の構成を単純化して段数を増やす方が，応用範囲が広がると考えられるので，各段に，LD/ST ユニット × 1，整数演算器 × 3，分岐ユニット × 1，メディア演算器 × 4 の合計 9 演算器を備える構成を仮定する[*5]．L1 キャッシュは，通常 VLIW 動作時には 4 ウェイセットアソシアティブ，アレイ動作時には 4 ブロックのプリフェッチ/Dirty 書き出しバッファとして機能する．L0 キャッシュも 4 ブロック構成と

[*4] 現在は手動で挿入しているが，自動挿入は今後の重要な課題である．
[*5] 本稿では，画像処理を対象としているためこのような構成としたが，たとえば科学技術計算への応用を考える場合には，浮動小数点数レジスタと浮動小数点演算器を各段に搭載するといった変更が必要である．

表 1　RTL シミュレータの諸元
Table 1　Parameters of RTL simulator.

初段	
命令デコード幅	最大 8 命令/cycle
汎用レジスタ（GR）数	32
メディアレジスタ（MR）数	32
外部バスとの転送速度	8 bytes/cycle
命令キャッシュ	4way 16 KB（64 bytes/line）
L1 キャッシュ	4way 16 KB（64 bytes/line）
L1 ⇒ L0 キャッシュ転送	12 bytes/cycle
ストアバッファ	4entry
全 30 段	
命令デコード結果設定	2 cycle/段
L0 キャッシュ	4block 128 B （各 block は 16 bytes/line×2）
L0 ⇒ L0 キャッシュ伝搬	12 bytes/cycle
L0 ⇒ LD 転送速度	4 bytes/cycle
ストアバッファ	1entry
最終段から L1 への転送	4 bytes/cycle

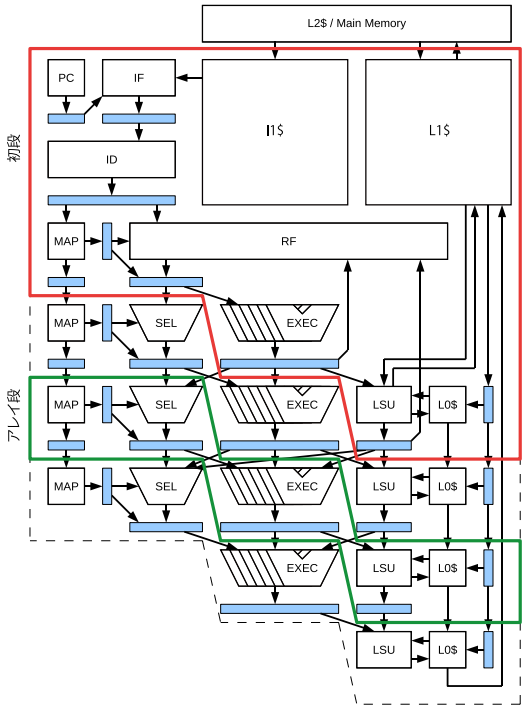

図 5　線形アレイアクセラレータの構成
Fig. 5　Block diagram of a linear array accelerator.

する．16 バイトにより 1 ラインを構成し，2 ラインにより 1 ブロックを構成する．これは，3 × 3 の画素が 16 バイト境界をまたがる場合にも全画素を保持するためである．

提案アクセラレータの構成を**図 5** に示す．これは 4 段構成の例である．ここで PC はプログラムカウンタの更新，IF は命令フェッチ，ID は命令デコード，RF はレジスタファイル，I1$は命令 1 次キャッシュ，L1$はデータ 1 次キャッシュ，EXEC は演算器群，LSU はロード/ストア機構，MAP は命令マップ，SEL はアレイ段間のセレクタ，L0$は L0 キャッシュである．

表 2　測定結果
Table 2　Measurement results.

	速度比	平均 IPC	最大 IPC	デコーダ動作率	段数	サイクル/画素
Z1	1.0	2.1	-	1.00	-	34.3
Z2	16.2	34.8	76	0.24	29	2.1
S1	1.0	2.6	-	1.00	-	20.8
S2	7.4	22.0	62	0.08	27	2.8
M1	1.0	2.9	-	1.00	-	16.1
M2	5.9	18.2	50	0.06	23	2.7
E1	1.0	2.3	-	1.00	-	6.8
E2	2.8	9.5	23	0.05	14	2.4
N1	1.0	2.5	-	1.00	-	6.6
N2	8.0	21.6	71	0.14	26	0.83

※ 1 画素 4 バイト．N のみ 1 画素 1 バイト．

5.1　性能評価

以上の条件で，前述の（1）と（2）の速度比，各フィルタ実行中の平均 IPC，アレイ動作中の最大 IPC，デコーダ動作率，1 画素あたりの実行サイクル数の測定を行った．結果を**表 2** に示す．なお，（1）と（2）では適用した最適化手法の違いから実行プログラムが異なるので，IPC の比率と速度比はかならずしも一致しない．

速度比と IPC について　速度比の結果から，アレイ動作は通常 VLIW 動作の 2.8〜16.2 倍の性能を発揮できることが分かる．（2）の結果において，多くのベンチマークの平均 IPC が，一斉演算中に観測される最大 IPC の 3 分の 1 程度であるのは，4 章に述べた L1 プリフェッチと Dirty 追い出しのオーバヘッドのためである．

デコーダ動作率について　デコーダ動作率を比較すると，(1) では 100%動作するのに対して，(2) では 5〜24%程度に抑えられている．L1 プリフェッチ/Dirty 追い出し時，および，一斉演算中は，デコーダとともに命令キャッシュも停止できることから，消費電力削減の効果が期待できる．本評価では横 320 画素の画像データを入力として用いたが，さらに大きな画像を対象とすると，デコーダ動作率はさらに減少する．Z2 では拡大処理であり，入力データのプリフェッチ時間が短く，相対的にデコーダ動作率が大きくなっている．

段数について　評価に用いたプログラムの場合，30 段あれば十分であることが分かった．今後，さらにプログラムの種類を増やして評価を進める．

サイクル/画素について　（1）の結果が大きくばらつき，プログラムの処理内容によって実行速度が大きく変化するのに対して，（2）では N を除くすべてのプログラムにおいて 2〜3 サイクル/画素と安定したスループット

表 3　回路規模
Table 3　Area estimation result.

	回路規模 [gates]	実装ユニット数 既存プロセッサ	初段	アレイ段
ユニット				
PC	685	1	1	0
IF	42,908	1	1	0
ID	26,247	1	1	0
RF	87,388	1	1	0
I1$	161,005	1	1	0
L1$	258,220	1	1	0
EXEC				
EAG	4,396	1	1	1
ALU	10,522	3	3	3
MEDIA	5,809	4	4	4
BRC	4,744	1	1	1
LSU	11,345	1	1	1
MAP	23,961	0	1	1
SEL	24,896	0	0	1
L0$	22,820	0	1	1
合計				
既存プロセッサ	651,739			
初段	698,520			
各アレイ段	146,963			

を実現しており，提案アクセラレータが処理内容によらず安定した出力が可能であることが分かった．このことから，必然的に平均 IPC と段数が比例していることが確認できる．

N2 の値が他のプログラムに比べて小さいのは，1 画素あたり 4 バイトを入力とする他のプログラムに対し，N2 では 1 要素あたり 1 バイトの輪郭情報を入力とし，1 サイクルで 4 要素分を 1 度に出力できるからであり，測定結果も約 1/4 の値になっている．

5.2　回路規模評価

提案アクセラレータの HDL 記述を 180 nm，1.8 V テクノロジ，120 MHz 制約において Design Compiler 2008.09 によって論理合成し，得られた cell area をゲート数に換算し回路規模を求めた．評価結果を**表 3** に示す．EXEC はさらに EAG：実行アドレス生成，ALU：算術演算器，MEDIA：メディア演算器，BRC：分岐を含み，各キャッシュにはコントローラ回路を含む．ユニットは各モジュールの回路規模，合計は各構成での回路規模である．**既存プロセッサ**にはプロセッサにはアレイ実行用の MAP，SEL，L0$は含まれず，**各アレイ段**には PC，IF，ID，RF といったフロントエンドや L1$，D1$は含まれない．一方，**初段**には SEL 以外のすべてが含まれる．

まず，L1$と I1$の回路規模を比較すると容量が同一であるにもかかわらず，L1$の方が約 1.6 倍大きくなっている．

表 4　回路規模比較
Table 4　Area estimations for each architecture.

	構成	回路規模 [gates]
既存プロセッサ	既存プロセッサ × 1 コア	651,739
メニイコア	既存プロセッサ × 7.6 コア	4,953,217
提案アクセラレータ	30 段（初段 + 29 アレイ段）	4,960,441

表 5　面積あたり性能
Table 5　Performance per area estimations.

ベンチマーク	既存プロセッサ	提案アクセラレータ*1	メニイコア *2	性能比 (*1/*2)
Z	7.6	16.2	7	2.31
S	7.6	7.4	7	1.06
M	7.6	5.9	7	0.84
E	7.6	2.8	7	0.40
N	7.6	8.0	7	1.14

これはポート構成の違いや，L1$には初段だけではなく最終段からの書き込みパスが存在するためと考えられる．

次に，提案アクセラレータと既存プロセッサおよびメニイコア（マルチコア）プロセッサの回路規模の比較結果を**表 4** に示す．この結果から，メニイコアのコア間ネットワークや同期機構の回路面積が 0.6 コア分の面積相当以下となり 7 コアのプロセッサが作成できたと仮定すると，30 段構成の提案アクセラレータは 7 コアのメニイコアと同等の回路規模であることが分かる．さらに，メニイコアのバス競合や同期オーバヘッドをいっさい無視し，1 コア時の 7 倍の性能が発揮できたと仮定すると，同一面積あたりの性能は**表 5** のようになる．この結果から提案アクセラレータは複雑なフィルタ処理に対して，理想的なメニイコアの最大 2.3 倍の性能が得られることが分かる．簡単な処理に対しては 1/2.5 程度の性能しか発揮できていないが，5.1 節で述べたとおり，処理の複雑さによらず安定した出力を実現した場合に，簡単な処理に対して性能が低下することは問題ではない．また，このような処理を実行している間には後半のアレイ段のような命令が割り当てられていない部分に対して省電力技術を適用することにより，消費電力の大幅な削減が期待できる．

6.　おわりに

本稿では，我々が提案しているプリフェッチ命令を挿入した VLIW 命令列を効率良く実行する線形アレイ VLIW アクセラレータにおいて，命令デコーダを流用してループ構造の全命令を各演算器に写像したうえで，命令デコーダを含むフロントエンドを停止し，既存プロセッサと互換性

のある機械語命令で記述されたイタレーションをオーバラップ実行するモデルを実装，評価した．その結果，最内ループのうち以下の条件を満たす場合に，ループの複雑さによらず毎サイクル1要素の安定した結果出力を実現した．

- 命令数が物理段数に収まる．
- イタレーション間に依存関係がない．
- ループ回数またはその上限が既知である．
- ロード/ストア命令のアクセスパターンが解析可能である．
- ループ中の演算結果のうち，ストア値以外の中間結果が不要である．

RTL シミュレータを用いた評価により，30 段構成の提案アクセラレータにおいて既存の VLIW 型プロセッサと比較して最大 16.2 倍，同一面積の理想的なメニイコアと比較しても最大 2.3 倍の性能を達成できることを示した．

今後は，RTL シミュレータをベースに HDL 記述を進め，FPGA での動作結果をもとにハードウェア量や遅延時間の評価を行うとともに，省電力技術を適用した場合の電力評価を行う予定である．また，高速実行可能なループがどの程度の割合で存在するのかを調査し，提案手法の適応範囲を明らかにすることや，プリフェッチ命令の自動挿入や演算器アレイのサイズを超えるループについての検討も，さらなる適応性の拡大のための重要な課題である．

謝辞　なお，本研究の一部は先端的低炭素化技術開発（次世代低電力デバイス安定化計算機構成方式）および科学研究費補助金（若手研究（B）課題番号 22700053）による．本研究は東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社，ローム株式会社，および日本ケイデンス株式会社の協力で行われたものである．

参考文献

[1] Vangal, S. et al.: An 80-Tile 1.28TFLOPS Network-on-Chip in 65 nm CMOS, *International Solid State Circuits Conference*, pp.98–99 (2007).
[2] Bell, S. et al.: Tile64 Processor: A 64-Core SoC with Mesh Interconnect, *International Solid State Circuits Conference*, pp.88–89 (2008).
[3] Kyo, S., Okazaki, S. and Arai, T.: An Integrated Memory Array Processor for Embedded Image Recognition Systems, *IEEE Trans. Comput.*, Vol.56, No.5, pp.622–634 (2007).
[4] Becker, J. and Hübner, M.: Run-time reconfigurabilility and other future trends, *The 19th annual symposium on Integrated circuits and systems design*, pp.9–11 (2006).
[5] Kazuhiro, Y., Takuya, I., Takashi, N., Jun, Y., Hajime, S. and Yasuhiko, N.: An Instruction Mapping Scheme for FU Array Accelerator, *IEICE Trans. Information and Systems*, Vol.94-D, No.2, pp.286–297 (2011).
[6] Shiota, T. et al.: A 51.2 GOPS, 1.0 GB/s-DMA Single-Chip Multi-Processor Integrating Quadruple 8-Way VLIW Processor, *International Solid State Circuits Conference*, pp.194–195 (2005).
[7] Bouwens, F.J. et al.: Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array, *Proc. 3rd international conference on High performance embedded architectures and compilers*, pp.66–81 (2008).
[8] Mei, B. et al.: Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study, *Proc. conference on Design, automation and test in Europe - Volume 2*, pp.1224–1229 (2004).
[9] Park, H., Park, Y. and Mahlke, S.: Polymorphic pipeline array: A flexible multicore accelerator with virtualized execution for mobile multimedia applications, *Micro-42: Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp.370–380 (2009).
[10] Goldstein, S.C., Schmit, H., Budiu, M., Cadambi, S., Moe, M. and Taylor, R.R.: PipeRench: A Reconfigurable Architecture and Compiler, *Computer*, Vol.33, pp.70–77 (2000).
[11] Tanigawa, K., Hada, R., Hironaka, T. and Kojima, A.: A Reconfigurable Processor PARS and its Compiler, *Proc. Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.91–100 (2007).
[12] Mehdipour, F., Honda, H., Kataoka, H., Inoue, K. and Murakami, K.: An Accelerator Based on Single-Flex Quantum Circuits for a High-Performance Reconfigurable Computer, *An Accelerator Based on Single-Flex Quantum Circuits for a High-Performance Reconfigurable Computer* (2009).
[13] 佐藤真琴：小面積・低消費電力を指向した動的再構成プロセッサ FE-GA，映像情報メディア学会誌：映像情報メディア，Vol.63, No.9, pp.1199–1201 (2009).
[14] 中田　尚，上利宗久，中島康彦：画像処理向け線形アレイ VLIW プロセッサ，先進的計算基盤システムシンポジウム SACSIS2009，pp.293–300 (2009).

中田 尚　（正会員）

2004 年豊橋技術科学大学大学院工学研究科情報工学専攻修士課程修了．2007 年同大学院工学研究科電子・情報工学専攻博士後期課程修了．同年奈良先端科学技術大学院大学情報科学研究科助教．博士（工学）．計算機アーキテクチャとシミュレーションに関する研究に従事．ACM 会員．

吉村 和浩　（学生会員）

2007 年龍谷大学理工学部電子情報学科卒業．2009 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，奈良先端科学技術大学院大学情報科学研究科博士後期課程在学中．計算機アーキテクチャとプロセッサ開発に興味を持つ．電子情報通信学会学生会員．

下岡 俊介　（学生会員）

2010 年広島市立大学情報科学部情報工学科卒業．現在，奈良先端科学技術大学院大学情報科学研究科博士前期課程在学中．計算機アーキテクチャに興味を持つ．

大上 俊　（学生会員）

2010 年立命館大学理工学部電気電子工学科卒業．現在，奈良先端科学技術大学院大学情報科学研究科博士前期課程在学中．計算機アーキテクチャに興味を持つ．電子情報通信学会学生会員．

デビセッティ ベンカタ ラマナヴィーン　（学生会員）

2010 年大阪電気通信大学電子工学科卒業．現在，奈良先端科学技術大学院大学情報科学研究科博士前期課程在学中．デジタル回路やハードウェア設計，計算機アーキテクチャに興味を持つ．電子情報通信学会学生会員．

中島 康彦　（正会員）

1986 年京都大学工学部情報工学科卒業．1988 年同大学院修士課程修了．同年富士通（株）入社．VLIW 型プロセッサ，命令エミュレーション，高速 CMOS 回路設計等に関する研究開発に従事．工学博士．1999 年京都大学総合情報メディアセンター助手．同年同大学大学院経済学研究科助教授，2006 年奈良先端科学技術大学院大学情報科学研究科教授（コンピューティング・アーキテクチャ講座担当），現在に至る．計算機アーキテクチャに興味を持つ．電子情報通信学会，IEEE-CS，ACM 各会員．