Vol.2009-ARC-186 No.10
Vol.2009-HPC-123 No.10
2009/12/1

線形アレイ VLIW プロセッサにおける適応性検討

中田　尚[†1]　中島康彦[†1]

近年の高性能組み込み機器のライフサイクル短縮と開発コストの増加に対する解決策として，我々は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 VLIW プロセッサを提案している．本プロセッサは，一般的な VLIW プロセッサを線形アレイ型に拡張した上で，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写像し，入力データを順次流し込むことにより，毎サイクル出力を得ることを特長としている．

本稿では，性能モデルを構築しメニィコアプロセッサやベクトル型プロセッサとの比較を行う．また，RTL レベルシミュレータを用いて性能モデルの正確さを評価した．評価の結果，線形アレイ VLIW プロセッサは性能モデルに極めて近い実効性能を達成した．また，さらなる高速化のためには演算とデータ転送のオーバラップ動作が重要であることがわかった．

Adaptability of A Linear Array VLIW Processor

Takashi Nakada[†1] and Yasuhiko Nakashima[†1]

Recently, the requirements for both low power and stable high performance have been increased rapidly. Under this consideration, we proposed a linear array VLIW processor that concatenates several VLIW processors in a linear fashion. In this architecture, instructions of most inner loop are sequenced into an ALU network, and the input data is supplied continuously, so that the output data is produced every clock cycle.

This paper gives an insight study of the possible performance gaining in the above architectures. Performance models of a many-core, vector, and the linear array VLIW processor have been built for comparison. The accuracy of the performance model of the linear array VLIW processor is verified by an RTL simulator. Our result shows the linear array VLIW processor can achieve a performance very close to the mathematical model.

1. はじめに

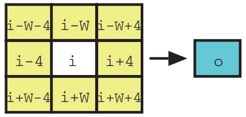
近年，組み込み機器が取り扱う情報量が増大し，低消費電力かつ最低性能を保証可能なプロセッサの需要が急速に高まっている．これまで，最低性能の保証は，専用ハードウェア化による実現が一般的であったものの，最近では，次々に策定される画像や無線等の新規格に追随したり，製品差別化のためのフィルタ微調整や出荷後の機能更新に対応するために，時間的・経済的コストが大きい専用ハードウェアの採用が難しくなってきている．しかし，入出力スループットが限定された条件下でも，定常的に結果を出力できるような専用ハードウェアに匹敵する性能を有し，かつ，汎用的な機械語命令を実行可能な高性能かつ柔軟なアーキテクチャは，現在のところ皆無である．

そこで我々は，低電力 ILP プロセッサの代表格である **VLIW**[1] と，軟らかいハードウェアの代表格である**リコンフィギャラブルデータパス**[2] を融合した線形アレイ VLIW プロセッサを提案している[3]．VLIW の立場からは飛躍的な高性能化，また，リコンフィギャラブルデータパスの立場からは，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の獲得という 2 つのブレークスルーがもたらされる可能性がある．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による高速性の両立を目指す仕組みに，ADRES[4,5] や TRIPS[6] がある．4×4 構成の場合の ADRES 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 程度の粗粒度リコンフィギャラブルアレイ（CGRA）を接続する．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．一方，TRIPS はデータフローとアレイ構造を直接記述できる EDGE 命令セット[7] を用いることで，多数の演算器を制御することを目指している．

一方，線形アレイ VLIW プロセッサでは，CGRA 命令や EDGE 命令セット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 VLIW 命令列をそのまま利用する．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**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 VLIW プロセッサ上でも実行できる**下位互換性** も備えている．

また，ADRES や TRIPS と異なり，アレイ動作中には，命令キャッシュを含むフロント

†1 奈良先端科学技術大学院大学
Nara Institute of Science and Technology




```
for (k=W-2; k>0; k--) {
  if (SAD(*(i-1),*(i+1))+SAD(*(i-W-1),*(i+W+1))
     +SAD(*(i-W),*(i+W))+SAD(*(i-W+1),*(i+W-1))>C)
    *o = 255;
  else
    *o = 0;
  i++; o++;
}
```

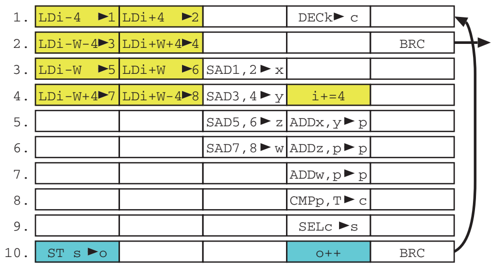


図 1 VLIW による輪郭抽出
Fig. 1 Edge detection written in VLIW.

エンド部分を完全に停止し，演算器アレイのみで VLIW 命令列と等価な演算を実行するとともに，演算器ネットワークについてもアレイ実行中は各演算器の演算の種類と相互接続関係は固定であるので，再構成のコストは低く抑えられるとともに，命令が割り当てられなかった演算器については，クロックゲーティング等の手法を用いることができる．以上の特徴により，消費電力を劇的に削減し超低電力コンピューティングを実現できる可能性もある．

本稿では，線形アレイ VLIW プロセッサにおける，プログラム記述における制限や，入出力スループットが限定された条件下での性能特性を明らかにすることにより，その適応性を明らかにする．

2. 線形アレイ VLIW プロセッサ

本章では，我々が提案している線形アレイ VLIW プロセッサの概要について述べる．

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，対角画素のバイト毎 SAD（Sum of Absolute Difference）の総和が閾値を越えた場合，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（BRC）はループカウンタ（k）が 0 になった際のループ脱出用，最後尾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．この命令列を既存の VLIW プロセッサで理想的に実行した場合，10 サイクル毎に 1 画素の輪郭情報を生成できる．

線形アレイ VLIW プロセッサでは，従来型 VLIW では演算器バイパスとして実現される

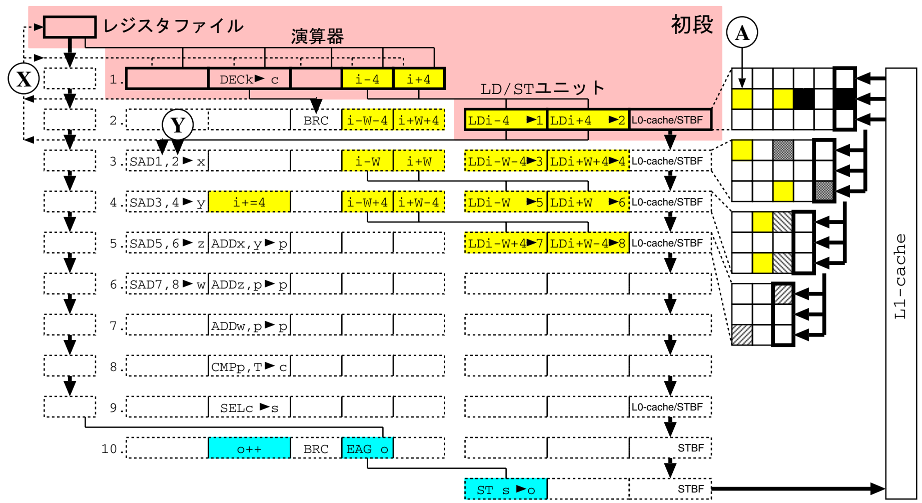


図 2 線形アレイ VLIW プロセッサ

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上で，命令デコーダを含むフロントエンドを停止する．ループの回転数に相当する入力データを主記憶から初段へ流し込むことにより，機械語命令レベルの互換性を保ったまま毎サイクル 1 画素の出力データ（輪郭情報）が得られる．

2.1 物 理 構 成

線形アレイ VLIW プロセッサの構成を**図 2** に示す．一般的な VLIW プロセッサを初段に配置し，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ている．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い．また，前述の ADRES 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，本提案では一方向に流れるデータが互いに干渉しないため，演算器ネットワークを見通し良く構築できる．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

Vol.2009-ARC-186 No.10
Vol.2009-HPC-123 No.10
2009/12/1

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）．

このプログラムを正しく実行するためには，順次実行される各段のロード命令が論理的にはすべての段で同じ内容のキャッシュにアクセスできなければならない．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．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，物理的には異なる L0 キャッシュに接続されているものの，論理的な内容は同一であり，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．したがって，ある特定の時刻においては，それぞれの段は世代の異なるイタレーションを実行するため，それぞれの段に付随するキャッシュは異なる世代のデータを保持している．隣り合った世代間の差分に注目するとその差はわずかであるため，差分を毎サイクル伝搬することにより論理的に正しい内容を保持する L0 キャッシュを実現できる．

本プロセッサの特長は，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．

2.2 制限事項

本節では，線形アレイ VLIW プロセッサ固有の制限事項について説明する．

- アレイ段数
  本アレイ構造では段数があらかじめ固定されており，アレイ中に収容可能な VLIW 命令列の長さが制限される．アレイ段数を超えた長さのループが入力されると，強制的にアレイ実行を中断し，通常の VLIW プロセッサとして動作する．したがって，長大ループを効率よく実行するためにはあらかじめループを分割するなどの工夫が必要である．
- 入力データサイズ
  アレイ実行を行うためには，ループ中でアクセスされるすべてのデータを L1 キャッシュにプリフェッチする必要がある．したがって，L1 キャッシュの容量よりも大きなデータを扱う場合にはループを分割するなどの工夫が必要である．同様に，1 つのイタレーションがアクセスするデータは，L0 キャッシュの容量以下である必要がある．
- ストアスループット
  各イタレーションではストアデータのサイズは 1 ワード以下に限定されている．1 ワードの範囲であればバイト単位のストアに分割されていても良いが，1 ワードを超えるデータを生成するループはやはり分割する必要がある．
- イタレーション間の依存関係
  アレイ実行では基本的に初段から最終段に向かってデータが流れるため，イタレーション間の依存関係に対応することは不可能である．ただし，ループ変数の更新やベースアドレス更新のような自己更新型の依存関係は利用できる．また，これを拡張し最大値・最小値を求める演算にも対応することができる．
  同様にメモリを介した依存関係についても非対応であり，アレイ実行中で書き込んだアドレスを以降のイタレーションで読み出してもアレイ開始前の値が得られる．
- アクセスレジスタ数
  論理的にはアレイ構造にマップされた命令はすべてのレジスタにアクセス可能である．しかし，アレイにマップされた時点で命令の種類やオペランドは確定されているため，すべての命令で必要となるレジスタはマップが完了した時点で確定する．このとき，以降の命令でアクセスされないレジスタ値は後段に伝搬する必要は無い．この性質を用いることにより，値の伝搬を最小限にすることができるとともに，各段の間に実装するレジスタの数を削減することができる．ただし，実装されているレジスタ数以上の伝搬が発生するループを実行することが不可能になるので，実装レジスタ数は慎重に決定しなければならない．
- アレイ実行後のレジスタ値
  前項で述べたとおり，アレイ実行中にレジスタを更新しても次段以降で利用されない場合にはそこで伝搬は終了し，最終段まで伝搬されることはない．したがって，最終的な値が初段のレジスタファイルに書き込まれることもない．このことは一般的なプロセッサで期待される動作と異なるために注意が必要である．最終的な値を初段のレジスタに反映させるためには，最終段から初段のレジスタファイルに書き込むパスを追加するとともに，アレイの途中で伝搬を中断した値をアレイ実行終了後に回収するフェーズが必要となる．
- 制御構造
  アレイ実行の対象となるのは最内ループのみであり，ループを継続する後方分岐とループを脱出する前方分岐の各 1 つが対象ループ中で記述可能な分岐命令である．ループ開始前に参照するすべてのデータをプリフェッチする必要があるので，ループ回転数の最大数をあらかじめ知る必要がある．ループの終了は前方分岐が成立した時であり，最大数に達する前に成立してもかまわない．

Vol.2009-ARC-186 No.10
Vol.2009-HPC-123 No.10
2009/12/1

アレイ対象ループの中に外側から飛び込んだ場合には，プリフェッチ命令が実行されないので通常のループとして実行される．また，if 文は条件付き実行で実現する．

以上の制限事項に違反した場合には期待した性能が得られないばかりでなく，間違った結果を出力する可能性がある．

2.3　ベクトル型プロセッサとの比較

線形アレイ VLIW プロセッサの制限はベクトル型プロセッサと多くの共通点があるが，専用の命令セットではなく，汎用の VLIW 命令セットを用いる点が大きく異なる．また，ベクトル型計算機では N 番目の演算はベクトルレジスタの N 番目の要素しか演算対象とできないが，線形アレイ VLIW プロセッサでは通常の LD/ST 命令でアクセスできるため，このような制限は存在しない．したがって，画像フィルタや差分法による数値シミュレーションのように，ある点とその近傍の値を用いて演算を行う場合に非常に有効である．

また，線形アレイ VLIW プロセッサではベクトル型プロセッサと比較して，より多くの演算器を搭載することが可能である．ベクトル型プロセッサでは演算器を増やすためにはベクトルレジスタの数およびその入力ポート数を増やす必要があり，実装が急激に困難になってしまう．一方アレイ構造ではアレイ段数を増やすことにより演算器のコストを線形に抑えることができる．また，各段に伝搬用レジスタを分散配置していることにより入力ポート数の問題も発生せず，ハードウェア拡張のスケーラビリティに優れている．

3.　性能モデル

線形アレイ VLIW プロセッサの特徴は，既存プログラムの親和性だけではなくその性能にもある．プリフェッチを行い，ループ中でキャッシュミスが発生しないことを保証することにより，データを演算器アレイに連続的に供給する．演算器アレイではすべての演算内容と演算器ネットワークが固定的に定まっているため，入力されたデータは固定サイクル後に出力される．したがって，定常的には毎サイクル 1 つの結果が出力されることになる．このような性能をモデル化するとともにその優位性を明らかにするために，線形アレイ VLIW プロセッサに加えて，一般のメニィコアプロセッサとベクトル型プロセッサについて，性能モデルを構築する．

本稿の目的は線形アレイ VLIW プロセッサの有効性を明らかにするためであるので，メニィコアおよびベクトル型プロセッサにおいては各種オーバヘッドを積極的に 0 と仮定し，議論を単純化する．

表 1　ハードウェア構成

共通	
メモリスループット	4 byte/cycle
メモリレイテンシ	0
線形アレイ VLIW プロセッサ	
アレイ段数	33
LD/ST ユニット数	1/段
ベクトル型プロセッサ	
ベクトル演算器数	無限
ベクトルレジスタ長	無限
ベクトルレジスタ数	無限
ベクトル LD/ST ユニット数	1
メニィコア	
コア数	無限

3.1　ハードウェア

本節では性能モデルに必要なハードウェアパラメータについて述べる．**表 2** に本稿で仮定したハードウェア構成を示す．

本稿では，入出力スループットを一定とした時のアレイとベクトルの性能をモデル化するため，メモリスループットはすべてのプロセッサにおいて 4 byte/cycle とした．一方メモリレイテンシはモデルを単純化するため 0 とした．

線形アレイ VLIW プロセッサにおいて，現在のシミュレータの実装では，アレイ段数を無限とすると，結果が無限に出力されなくなるため便宜的に上限を指定する．33 段はこれまでの予備調査により実用上十分と考えられる値である．

外部バスと直接接続されていない L0 キャッシュの機能により，ロード命令は各段で 1 つ実行可能で，同一の時刻に異なる世代のデータに対して同時にアクセスできる．ストア命令は出力スループットの制限により同一の時刻には 1 つのストアのみが実行できる．

線形アレイ VLIW プロセッサとメニィコアプロセッサのデータ 1 次キャッシュサイズは最内ループ中で必要となるデータが十分に格納できるだけの大きさとし，演算開始の前にすべてのデータをプリフェッチすることとする．ただし，プログラム全体でアクセスするデータすべてを格納することはできないとする．ベクトル型プロセッサでは通常通りベクトルロード（VLD）を用いてメインメモリからロードを行う．

3.2　ソフトウェア

ハードウェアと同様に，ベンチマークプログラムについてもいくつかのパラメータで表現

Vol.2009-ARC-186 No.10
Vol.2009-HPC-123 No.10
2009/12/1

する．モデル化は以下の方針で行うこととする．

- 関数単位に分割し，パラメータ化する．
  見通しを良くするため，関数単位でパラメータ化を行い，必要に応じて組み合わせることにより，アプリケーション全体の性能を予測する．
- イタレーション間の依存関係は無いアプリケーションを対象とする．
  イタレーション間に依存関係がある場合はどのプロセッサにおいても特別な配慮が必要であるため，本稿では対象としない．
- イタレーション内の依存関係は無い，または十分な命令レベル並列性があるものと仮定する．
  この仮定は非現実的であるが，線形アレイ VLIW プロセッサにおいては予備評価により，依存関係を考慮してもアレイ構造にマッピングが可能であれば同等の結果となることがわかっている．メニィコアとベクトル型プロセッサにおいては議論を単純にするため，理論値通りの性能が達成できると仮定する．

以上の方針により，ベンチマークプログラムのパラメータ化を行う．まずはじめに，入出力データ長すなわち最内ループの回転数を N とし，1 回のイタレーションに含まれる算術演算の数を OP とする．これには，アドレス計算や，ループの終了判定に必要な命令を含まないこととする．

次に，入出力データの単位としてストリームを定義する．ストリームとは長さが N の 1 次元配列とする．対象ループ内で必要となる入力用ストリームの数を S_{IN}，出力用ストリームの数を S_{OUT} とする．

対象ループの外側にループがある，すなわち 2 重以上のループである場合で，最内ループが終了した後に，再び最内ループの実行が行われる場合には，直前の最内ループで利用したストリームがキャッシュに残っている場合があり，新たにプリフェッチしなければいけないストリームの数は前述の S_{IN} よりも少なくなる可能性がある．このときのプリフェッチが必要なストリームの数を S_{NEW} とする．

ベクトル型プロセッサにおいて，最内ループ中で必要なベクトルロードの数はこれまで述べたストリームの数とは一般に異なり，ループ中で参照される配列型変数の数に等しい．例えば i を誘導変数とするループ中で `y[i] = x[i] + x[i+1]` のような式を実行する場合の，入力用ストリーム数 S_{IN} は 1 であるが，ベクトル型プロセッサにおいて，必要な入力用ベクトルレジスタの数は 2 であり，必要なベクトルロードの数も 2 である．この値を LD で表す．ストリームと同様に 2 重以上のループの内側ループである場合には，ベクトルレジスタについても再利用が可能であるので，この時に必要なベクトルロードの数を LD_{NEW} とする．

出力についても同様に，出力ストリーム数を S_{OUT}，必要なベクトルストアの数を ST とする．

表 2 にソースコードの分析によって得られた各種パラメータの値を示す．ここで filter:edge は図 1 で示した輪郭抽出フィルタであり，3×3 を単位として SAD（絶対値差分総和）演算を適用することで輪郭情報を出力する．SPECfp ベンチマークからは swim と mgrid を対象とした．それぞれのベンチマークプログラムから主要な関数として，swim では calc1 と calc2，mgrid からは resid 関数を対象とした．ここで，resid は $3 \times 3 \times 3$ の立方体中の 27 要素の値から 1 つの出力を得るため，入力に関するパラメータが比較的大きくなっている．Linpack ベンチマークからは主要な関数である dgefa 関数と dgesl 関数を対象とした．さらにこれらの関数からは daxpy，最大値演算を必要とする idamax，ベクトルスカラ積を行う dscal が呼び出される．これらのうち idamax と dscal 関数は入力データに再利用性が無く，S_{IN} と S_{NEW}，LD と LD_{NEW} の値が等しくなっている．

表 2 ソフトウェアのパラメータ

Benchmarks	OP	S_{IN}	S_{NEW}	LD	LD_{NEW}	S_{OUT}	ST
filter:edge	18	3	1	9	3	1	1
swim:calc1	23	6	3	10	6	4	4
swim:calc2	26	11	7	16	11	3	3
mgrid:resid	30	9	3	27	9	1	1
linpack:daxpy	2	2	1	2	1	1	1
linpack:idamax	1	1	1	1	1	1	1
linpack:dscal	1	1	1	1	1	1	1

3.3 性能モデル

これまで定義したパラメータを用いて，各プロセッサの性能モデルを決定する．あるプログラムを実行した時の線形アレイ VLIW プロセッサ，メニィコアプロセッサ，ベクトル型プロセッサの実行サイクル数をそれぞれ T_A，T_M，T_V とする．

図 3 に filter:edge を各プロセッサで実行した場合の最も理想的な動作パターンを示す．

- 線形アレイ VLIW プロセッサ
  図 3(a) に示すように，プリフェッチに $S_{NEW} \cdot N$ サイクル必要であり，演算には N サイクルに加えてアレイ段数に比例するオーバヘッド $ArrayOverhead$ がかかる．また，1 回のアレイ実行中に書き込めるストリームは 1 つだけであるので，出力用ストリーム

Vol.2009-ARC-186 No.10
Vol.2009-HPC-123 No.10
2009/12/1

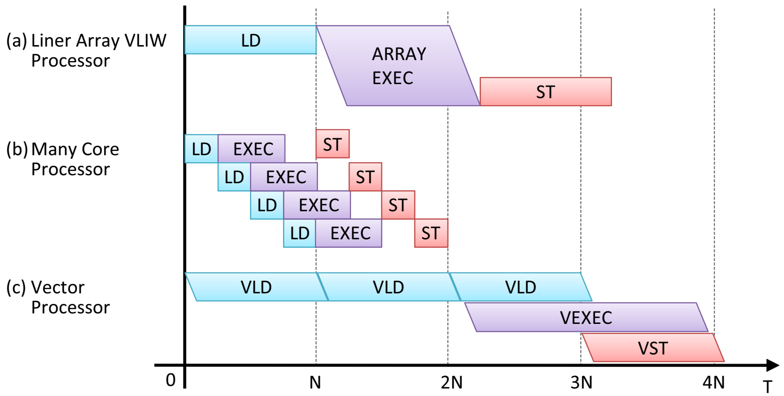


図 3 各プロセッサの実行の流れ

の数だけアレイ実行を繰り返す必要がある．したがって実行サイクル数 T_A は次のようになる．

$$T_A = (S_{NEW} + S_{OUT}) \cdot N + (N + ArrayOverhead) \cdot ST$$
$$= (S_{NEW} + S_{OUT} + ST) \cdot N + ArrayOverhead \cdot ST$$

プロセッサ中の演算器の合計は各段の演算器数と段数の積となるが，少なくとも OP 個の演算器が必要であることは明らかである．

- メニィコアプロセッサ

図 3(b) に示すように，入出力スループットが限定された状況では全体で 1 つのコアのみがロード命令を完了できる．コア数が十分多ければ，最後のコアがロードを完了した時点で 1 番目のコアがストアを開始できる．したがって実行サイクル数 T_M は次のようになる．

$$T_M = (S_{NEW} + S_{OUT}) \cdot N$$

このときのコア数を C とすると，1 コアあたりの演算時間は $N \cdot (OP/IPC)/C$ となり[⋆1]，この時間が残りのコアのロード時間 $N \cdot (C-1)/C$ と等しい時に必要コア数が最小となる．ここで IPC は演算中の平均 IPC であり，このときのコア数は $OP/IPC+1$ となり，1 コアあたり少なくとも IPC 個の演算器が必要であるので，プロセッサ中の演算器の合計は $OP + IPC$ 以上が必要となる．

- ベクトル型プロセッサ

図 3(c) に示すように，LD_{NEW} 回の VLD を逐次実行し，その後ベクトル演算器を用いた演算を行い，最後に結果を ST 回の VST で書き出す[⋆2]．ベクトルオーバヘッドの隠蔽と演算のスケジュールが理想的に行われた時には最後の VLD の 1 番目の要素が到着した時点からベクトル演算を開始し，VST の最後の要素が格納される前に演算が完了すればよい．したがってベクトルオーバヘッドを $VectorOverhead$ とすると，実行サイクル数 T_V は次のようになる．

$$T_V = (LD_{NEW} + ST) \cdot N + VectorOverhead$$

このときに必要なベクトル演算器の数は $OP/(ST+1)$ となる．

図 3 および以上の 3 式から，メニィコアプロセッサとベクトル型プロセッサにおいては理想的な条件下では，演算量によらずほぼデータ入出力時間によってのみ実行時間が決まることがわかる．線形アレイ VLIW プロセッサにおいてはこれに加えて演算時間が必要となっているが，この時間も命令数 OP とは独立である．

4. 評 価

本章では，これまでに述べたソフトウェアおよびハードウェアのモデルを用いて性能を予測するとともに，RTL レベルシミュレータを用いて実際の性能を測定し性能モデルおよび線形アレイ VLIW プロセッサの評価を行う．

4.1 評価条件

性能モデルの妥当性を評価するために，線形アレイ VLIW プロセッサの実行サイクル数を RTL レベルシミュレータを用いて測定した．シミュレーション対象のプロセッサの構成を**表 3** に示す．

以降では Linpack ベンチマークの dgefa 関数と dgesl 関数および SPEC ベンチマーク swim の calc1 関数と calc2 関数を対象とした評価について述べる． また，性能モデルは厳密にシミュレーション結果と一致するわけではないので，低次の項は適宜省略する．

4.2 性能モデルによる予測

linpack:dgefa

関数内では idamax，dscal，daxpy が呼び出される．それぞれの呼び出し回数は，idamax が長さ N から 1 までの N 回，dscal が $N-1$ から 1 までの $N-1$ 回，daxpy が長さ N

⋆1⋆2 部分演算に必要なデータが揃った時点で演算を開始することもできるが，簡単化のためここでは考えないこととする．また，そのように仮定してもここで求める実行サイクル数に影響はない．

表 3 RTL シミュレータの諸元

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	4bytes/cycle
命令キャッシュ	4way 16KB (64bytes/line)
L1 キャッシュ	10way 80KB (64bytes/line)
L1 ⇒ L0 キャッシュ転送	12bytes/cycle
ストアバッファ	4entry
全 32 段	
L0 キャッシュ	4block 128B (各 block は 16bytes/line×2)
L0 ⇒ L0 キャッシュ伝搬	12bytes/cycle
L0 ⇒ LD 転送速度	4bytes/cycle
ストアバッファ	1entry
最終段から L1 への転送	4bytes/cycle

から 1 までを N から 1 回ずつである．したがって，アレイ実行における総演算サイクル数 $EXEC_{dgefa}$ は次のようになる．

$$\begin{aligned}EXEC_{dgefa} &= \sum_{i=1}^{N}(i + ArrayOverhead) + \sum_{i=1}^{N-1}(i + ArrayOverhead) + \\ &\quad \sum_{i=1}^{N-1}\{i(i + ArrayOverhead)\} \\ &= (N^2/2 + N/2) + ArrayOverhead \cdot N + \\ &\quad (N^2/2 - N/2) + ArrayOverhead \cdot (N-1) + \\ &\quad (N^3/3 - N^2/2 + N/6) + ArrayOverhead \cdot (N^2/2 - N/2) \\ &= N^3/3 + (1/2 + ArrayOverhead/2)N^2 + O(N)\end{aligned}$$

また，プリフェッチおよびストアに必要なサイクル数 MEM_{dgefa} は，daxpy の S_{NEW} が 1 であることから次のようになる．

$$MEM_{dgefa} = N^3/3 + O(N^2)$$

linpack:dgesl

関数内では daxpy が長さ $N-1$ から 1 まで 2 回ずつ呼び出されるので，総演算サイクル数 $EXEC_{dgesl}$ およびメモリアクセスサイクル数 MEM_{dgesl} は次のようになる．

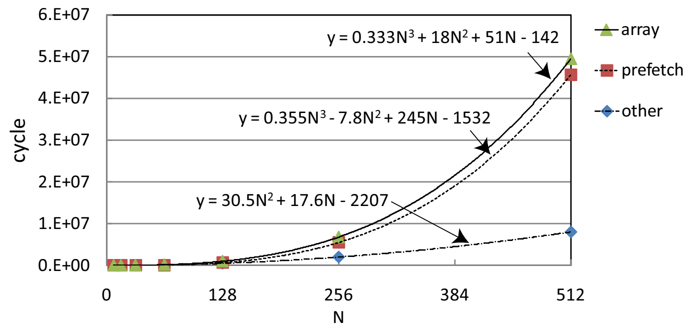


図 4 実行サイクル数（Linpack:dgefa）

$$EXEC_{dgesl} = N^2 + (2ArrayOverhead - 1) \cdot N + O(1)$$
$$MEM_{dgesl} = N^2 + O(N)$$

swim:calc1

2 重ループを含む関数であるので，その総演算サイクル数 $EXEC_{calc1}$ およびメモリアクセスサイクル数 MEM_{calc1} は次のようになる．

$$EXEC_{calc1} = 4N^2 + 4ArrayOverhead \cdot N$$
$$\begin{aligned}MEM_{calc1} &= (3+4)N^2 + O(N) \\ &= 7N^2 + O(N)\end{aligned}$$

swim:calc2

同様に $EXEC_{calc2}$ および MEM_{calc2} は次のようになる．

$$EXEC_{calc2} = 3N^2 + 3ArrayOverhead \cdot N$$
$$\begin{aligned}MEM_{calc2} &= (7+3)N^2 + O(N) \\ &= 10N^2 + O(N)\end{aligned}$$

4.3 シミュレータによる測定

RTL レベルシミュレータで問題サイズ N を変化させて実行を行い，実行サイクル数の内訳を測定した．結果を**図 4** から**図 7** に示す．ここで array がアレイ実行のサイクル数，prefetch がプリフェッチのサイクル数，other はそれ以外の外側ループやその他の命令を実行するサイクル数である．また，プロットが測定値であり，曲線と数式は最小二乗法による近似曲線とその式である．

まずアレイ実行のサイクル数（array）に注目すると，すべての結果において前節で求めた

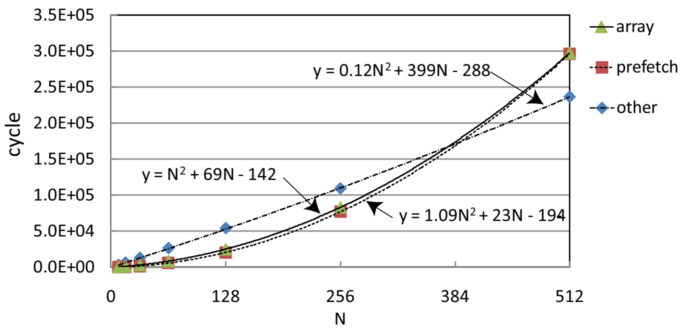


図 5 実行サイクル数（Linpack:dgesl）

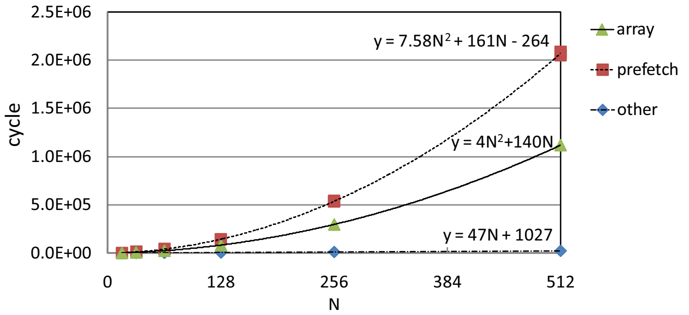


図 6 実行サイクル数（swim:calc1）

予測演算サイクル数と非常に近いことがわかる．特に主項は完全に一致している．第 2 項については性能モデルには $ArrayOverhead$ が含まれているが，近似曲線からこの値を求めるとすべての測定結果で $ArrayOverhead = 35$ という結果が得られる．今回のシミュレーションではアレイ段数を 33 段としているため，初段から最終段にデータが単に伝搬するだけでも 33 サイクル必要となるので，この値は妥当な結果といえる．

プリフェッチのサイクル数（prefetch）に注目すると，測定値と予測値はほぼ一致した結果が得られている．主項の誤差は 6.4%から 16.3%となっている．これはメモリ転送のオーバヘッドが性能モデルで考慮されていないことが原因と考えられる．

その他の処理にかかったサイクル数（other）は，主項がアレイ実行やプリフェッチと比較して次数が 1 次低いか，係数が 8 分の 1 以下になっており，十分に小さくなっていることがわかる．

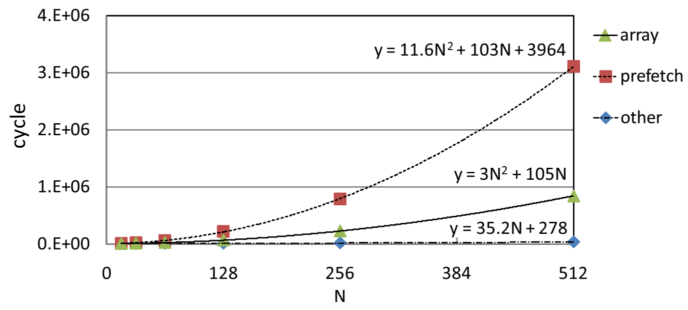


図 7 実行サイクル数（swim:calc2）

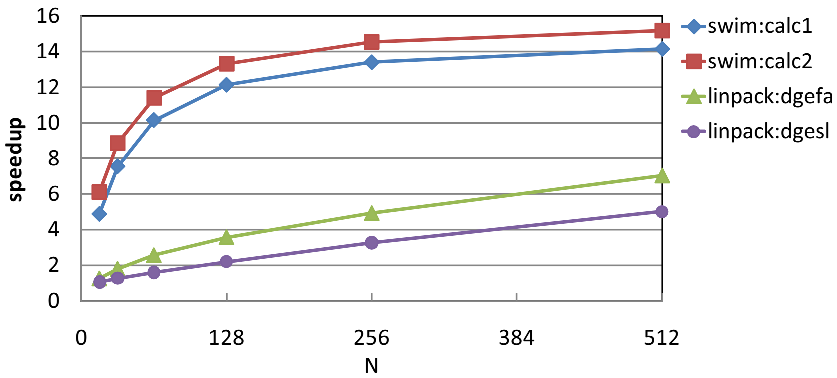


図 8 アレイ実行の高速化率

図 8 にアレイ実行を行わなかった場合と比較した時の，アレイ実行の高速化率を示す．測定は RTL レベルシミュレータで行い，アレイ実行以外のパラメータは同一とした．また，各ベンチマークプログラムの実行ファイルは両者の実行で全く同一である．

この結果から，問題サイズ N が大きくなるにつれ，アレイ実行のオーバヘッドやそれ以外の実行時間の占める割合が減少し，高速化率が向上していることがわかる．

Vol.2009-ARC-186 No.10
Vol.2009-HPC-123 No.10
2009/12/1

5. 考　　察

5.1 性能モデル

メニィコアプロセッサやベクトル型プロセッサの性能モデルとの比較から，線形アレイVLIWプロセッサでは演算とデータ転送が同時に行えないことが性能向上の妨げとなっていることが明らかになった．

例えば，次のアレイ実行が予測可能かつ，L1キャッシュに1ストリーム分の空きが確保できるのであれば，アレイ実行とオーバラップして次のアレイ実行のためのプリフェッチを行うことができる．この機能を実装すれば，アレイ実行のサイクル数 T_A は次のように改善される．

$$\begin{aligned} T_A &= (S_{NEW} - ST + S_{OUT} + ST) \cdot N + ArrayOverhead \cdot ST \\ &= (S_{NEW} + S_{OUT}) \cdot N + ArrayOverhead \cdot ST \end{aligned}$$

依然としてアレイオーバヘッドの項は残るが，4.3節で述べたとおり，$ArrayOverhead$ はアレイ段数程度であるので，N が十分大きければ無視できる．また，本稿ではメニィコアプロセッサやベクトル型プロセッサに対するオーバヘッドを無視したが，一般的にこれらのオーバヘッドは無視できない程度に大きいので，実効性能では線形アレイVLIWプロセッサの性能がこれらの性能を上回る可能性は十分にあると考える．

5.2 シミュレーションによる評価

RTLレベルシミュレータを用いた評価の結果，アレイ実行においてはモデルに非常に近い性能が発揮でき，本プロセッサの特徴である安定した性能が確認できた．

また，メモリアクセスサイクル数は6.4%から16.3%程度のオーバヘッドが確認できた．これはデータ転送がライン単位で行われるため，不要なデータ転送が行われたことや，メモリアクセス要求の処理にかかる時間が性能モデルでは無視されていることが原因と考えられる．

その他のサイクル数はアレイ実行やプリフェッチの実行時間と比較して十分小さく，特に問題サイズ N が十分大きければ無視できる程度の時間であり，$N = 512$ のときにswim:calc2で最大高速化率15.2倍を達成した．

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プロセッサの特性と性能モデルをまとめ，その適応性を明らかにした．実行可能なプログラムにはいくつかの制限があるが，それはベクトル型プロセッサと同等の制限がほとんどであり，実用上の問題は小さい．性能モデルを構築しメニィコアプロセッサやベクトル型プロセッサと比較したところ，線形アレイVLIWプロセッサには計算とデータ転送が同時に行えないことが性能向上を阻害していることがわかった．一方，RTLシミュレーションによる評価においては予測モデルに非常に近い性能が得られその安定した性能を確認することができた．演算とデータ転送のオーバラップを実装すればさらなる高速化が得られることが期待できる．

今後の課題としては，アレイ長を超えた場合の実行やプリフェッチ命令の自動挿入手法について検討を行う必要がある．また，RTLシミュレータをベースにHDL記述を進め，FPGAでの動作結果を元にハードウェア量や遅延時間，さらには消費電力の評価を進める予定である．

参　考　文　献

1) Shiota, T. et al.: A 51.2GOPS, 1.0GB/s-DMA Single-Chip Multi-Processor Integrating Quadruple 8-Way VLIW Processor, *ISSCC*, pp.194–195 (2005).
2) 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).
3) 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).
4) Bouwens, F.J. et al.: Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array, *HiPEAC'08*, pp.66–81 (2008).
5) Mei, B. et al.: Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study, *DATE*, pp.1224–1229 (2004).
6) Sankaralingam, K. et al.: Distributed Microarchitectural Protocols in the TRIPS Prototype Processor, *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, pp.480–491 (2006).
7) Burger, D. et al.: Scaling to the End of Silicon with EDGE Architectures, *Computer*, Vol.37, No.7, pp.44–55 (2004).