Vol.2011-ARC-196 No.18
2011/7/28

演算器アレイ型アクセラレータにおけるローカルバッファの最適化

下岡俊介[†1]　吉村和浩[†1]
中田尚[†1]　中島康彦[†1]

我々は，高電力効率かつバイナリ互換性を備えた演算器アレイ型アクセラレータ（LAPP）を提案している．LAPP は既存 VLIW 命令で記述されたプログラムの最内ループを演算器アレイに写像し，高速実行する．高速実行時は，写像されたロード命令はアドレス計算とタグ比較によりローカルバッファの該当 way からデータを読み出すことで実行される．ここで各ロード命令とローカルバッファのいずれかの way とは 1 対 1 対応していることを利用することで，各段のロードストアユニットに含まれるアドレス計算とタグ比較の論理を削減し，ローカルバッファに使用される値のみを保持するよう最適化することができる．本稿では，写像されるロード命令とローカルバッファの way を対応させ，各段のロードストアユニットのアドレス計算とタグ比較を削減すること，および，レジスタに置き換えることによりローカルバッファを最適化する手法を提案する．提案手法を HDL により実装し，回路規模および遅延時間を評価した．評価の結果，ローカルバッファの要素が 64 ワードまでは，ロードのレイテンシを 2 から 1 へ確実に削減可能であることが判明した．

Optimizing Local Buffers for FU Array Accelerator

Shunsuke Shitaoka,[†1] Kazuhiro Yoshimura,[†1]
Takashi Nakada[†1] and Yasuhiko Nakashima[†1]

Our previously proposed FU (functional unit) array accelerator LAPP, can achieve extemely high energy-efficiency by fully exploiting parallelism between inner loop iterations and using minimum necessary units to perform the calculation. For the acceleration purpose, an n-way local buffer is additionally attached to each array stage to efficiently supply data into the arrayed FUs. The load instruction is executed by indexing from the address calculation and comparing the request address with the tags from the indexed lines in all ways of local buffers. In order to reduce the unnecessary comparison, in this paper, we propose an optimized method by removing address calculation and tag comparison. We evaluated circuit area and delay of local buffers of the proposed method by an HDL implementation. The results have indicagted that the load latency can be reduced from 2 cycles to 1 cycle in the local buffers which contain 64 words per each way.

1. はじめに

近年，低消費電力かつ高性能なアーキテクチャが求められており，Larrabee[1] などのメニコアや ADRES[2]，TRIPS[3]，PPA[4] などの粗粒度リコンフィギャラブルアレイ（CGRA）が研究されている．メニコアは汎用プロセッサを多数並べた構成をとっており，並列処理できる状況であれば単一のプロセッサよりもプログラムを高速実行でき，既存プログラムとの互換性も保っている．しかし，電力あたり性能は単一のプロセッサと同等である．一方，CGRA は演算器を 2 次元アレイ状に並べた構成をとっており，専用ハードウェアに近い構造をしているため，メニコアよりも電力効率に優れる．しかし演算器間のデータパスを構成するために専用命令セットを用いており，既存プログラムとのバイナリ互換性がなく，専用コンパイラの開発が必須となる．我々は演算器を 2 次元アレイ状に並べた構成をとり，かつ既存機械語命令を使用することで既存プログラムとのバイナリ互換性も保つ，演算器アレイ型アクセラレータ LAPP（Linear Array Pipeline Processor）を提案している[5]．LAPP は従来の VLIW プロセッサのバックエンド部を拡張し複数段に渡って接続した構成をとる．そして，最内ループ内の命令（ループカーネル）を演算器アレイに写像し，アレイ実行させることで，高性能を実現している．また，アレイ実行中は未使用ユニットに長期間パワーゲーティングを適用することで，高い電力効率を実現している．さらに，既存 VLIW プロセッサの命令セットを一部拡張しているだけのため，バイナリ互換性も備えている．

従来は，アレイ実行中は必要なデータを L1 データキャッシュの各 way から，L1 データキャッシュよりも小規模なローカルバッファにデータを流しこみ，ローカルバッファの各 way に順次，一時的にデータが保持される．各演算器に写像されたロード命令は，タグ比較によりローカルバッファの該当 way からデータを読み出す．way に流し込むデータは連続データであり，アレイ実行中は単一のロード命令に注目すると連続アクセスである．一方，個々のロード命令は単一の way にしかアクセスしないため，各段のロードストアユニットのタ

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

グ比較とアドレス計算を簡素化し，ローカルバッファに使用される値のみを保持させることができると考えられる．データプリフェッチ命令によってロード命令によるアクセス範囲が指示され，ある 1 つの way に必ずヒットすることが保証できると考えられるため，タグ比較は不要となり，ダイレクトマップキャッシュのごとくインデックスと 1 対 1 に対応する要素を選択するだけでよい．本稿では，こうしたローカルバッファの不要な要素を削減するため，あらかじめ命令写像時にロード命令とローカルバッファの way を対応させることでローカルバッファを最適化する手法を提案する．最適化手法は一定の範囲のデータにランダムアクセスできる従来の能力を保ちつつ，ローカルバッファを削減することで回路規模，および回路遅延の面から最適化を行う．2 章では従来型 LAPP の概要と従来型 LAPP においてのローカルバッファについての問題点を述べる．3 章ではローカルバッファの最適化手法を詳述する．4 章では評価方法と評価結果について論じ，5 章でまとめる．

2. LAPP

本章では LAPP の構造，実行モデル，データプリフェッチとアレイ実行中のローカルバッファに対するアクセスの詳細について述べる．

2.1 LAPP の構造

図 1 に LAPP の構造を示す．LAPP の構造は大きく分割すると初段と，複数のアレイ段からなるアレイ部から成り，図 1 はアレイ 4 段構成の例である．段数は設計時に実行対象のプログラムが必要な段数を考慮して決定する必要がある．初段は従来型 VLIW プロセッサと同様の構成であり，プログラムカウンタ（PC），命令フェッチ（IF），命令デコーダ（ID），レジスタファイル（RF），演算器（EXEC），アドレス演算器（EAG）ロード/ストアユニット（LSU），L1 データキャッシュ（L1$），命令キャッシュ（I1$）を含む．アレイ部は従来型 VLIW プロセッサのバックエンド部であるアレイ段を線形に接続した構造をしている．アレイ段は演算器，アドレス演算器，ロード/ストアユニット，L0 伝搬レジスタ，セレクタ（SEL），伝搬レジスタ，ローカルバッファ（L0$）および命令マッパ（MAP）から成る．アレイ段はレジスタファイルに代替して伝搬レジスタとセレクタを備え，L1 データキャッシュに代替してローカルバッファを備えている．また，後段のローカルバッファに L1 データキャッシュからのデータ供給を行うために，L0 伝搬レジスタを備える．

2.2 実行モデル

LAPP は，通常実行モード，アレイ設定モード，およびアレイ実行モードの 3 つのモードを備える．通常実行モードでは，初段のみが動作し，アレイ段は停止する．データプリフェッチ命令のデコードを契機にアレイ設定モードに遷移する．アレイ設定モードでは，データプリフェッチ命令の次の命令からループの先頭に戻る無条件分岐命令までをループカーネルとみなし，ループカーネルすべての VLIW 命令列を演算器に写像し，演算器ネットワークを設定する．ループカウンタを更新する自己更新命令を演算器に写像するには，自身の出力を自身の入力へとバイパスする自己ループを設定する．命令写像については文献 6) を参照されたい．命令写像と同時に，データプリフェッチ命令の情報を基にアレイ実行で使用するデータを L2 キャッシュおよび主記憶からプリフェッチする．命令写像およびデータプリフェッチが完了次第，アレイ実行モードに遷移する．アレイ実行モードでは，プログラムカウンタ，命令フェッチ，命令デコーダおよび未使用演算器はパワーゲーティングが適用され，停止する．そして，L1$から L0 伝搬レジスタへとデータが供給され，各段のローカルバッファへ保存されつつ各段の演算器によってループカーネルがアレイ実行される．このときアレイ中では複数のイタレーションがパイプライン処理で並列に実行される．すなわちアレイ

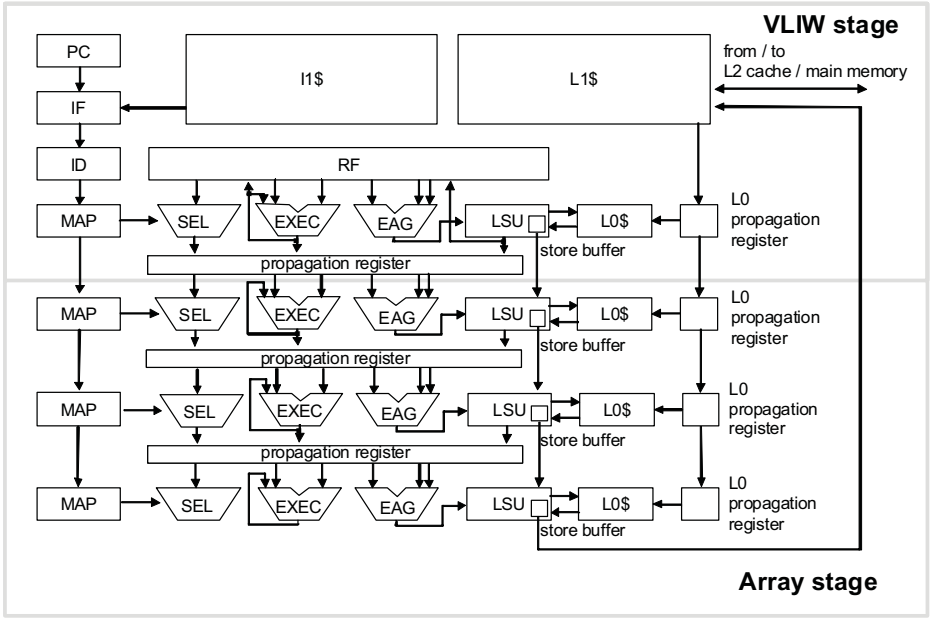

図 1 LAPP のモジュール構成

実行の i サイクル目のアレイ j 段目では（i−j+1）番目のイタレーションが実行される．各段での実行結果は伝搬レジスタを介して後段の演算器へと伝播され，演算結果のストアデータは最終段から L1$へと書き戻される．ループカーネル中のループ脱出用後方分岐命令の成立を契機にアレイ実行を終了し，通常実行モードに遷移する．

2.3 データプリフェッチとアレイ実行中のローカルバッファへのアクセス

アレイ設定におけるプリフェッチはデータプリフェッチ命令（DCPL: Data Cache Pre-Load）によるベースアドレス，プリフェッチ長，プリフェッチ対象の way，ストライド長，プリフェッチ方向の指定により L1$に対して行われる．プリフェッチはストリームプリフェッチ[7] と固定長のストライドプリフェッチ[7] が可能である．L1$に way は計 4 つ備えられており，読み書き専用の way0 と読み出し専用の way1，way2，way3 がある．読み書き専用である 1 つの way は通常実行においても使用するが，読み込み専用である 3 つの way はアレイ実行のためのプリフェッチバッファとして使用する．アレイ実行では，L1$の各 way から初段の L0 伝搬レジスタへ 1 ワードずつデータを転送する．データ供給は各 way から同時に行われるので，L0 伝搬レジスタは way の数と同じワード数である 4 ワード分のデータを保持する．

図 2 に従来型ロードストアユニットの回路図を示す．L0 伝搬レジスタに到着したデータは L0 伝搬レジスタを保持する段のローカルバッファと，次段の L0 伝搬レジスタへと転送される．ローカルバッファは L1 データキャッシュと同様 4 つの way から成り，L0 伝搬レジスタから対応する各 way にデータを転送し，ローカルバッファに保存する．ただし，ローカルバッファの容量は L1 データキャッシュと比較すると小規模で，伝搬されたデータを各 way において数ワード分のみ保持可能であるため，直前数サイクルに L0 伝搬レジスタから転送されたデータのみにアクセス可能であるが，データ供給タイミングと前述のパイプライン動作による実行タイミングを一致させることにより，常に必要なデータがローカルバッファに存在することを保証する．また，ロード/ストア命令の実行は回路遅延縮小のため，アドレス計算の後にロード/ストア命令が実行される．従って，ロード/ストア命令の実行にはレイテンシが 2 必要である．アレイ実行時においてロード命令が実行されると，ローカルバッファのタグ比較によって対応する way と対応する要素の位置を検索し，マルチプレクサ（MUX）で要求するデータを選択する．ただし，我々が用意した画像処理ベンチマークでは，アレイ実行時に実行されるロード命令はそれぞれ 1 つの way に対応しているため，残りの 3 つの way のデータは不要である．しかし，ハードウェアでは各ロード命令がそれぞれどの way に対応しているかが不明なため，タグによってローカルバッファ内を検索せざるを得ない．そこで，ハードウェアまたはソフトウェアで対応する way を検出する方法がある．前者では既存機械語命令をそのまま流用可能であるが，追加ハードウェアによって電力効率を損ねてしまう問題があり，本稿では後者をとる．LAPP のアレイ実行におけるロード命令と way 番号は 1 対 1 対応していることに注目すると，タグ比較を行い，way を選択してデータをロードする論理を削減することができる．LAPP ユーザまたはコンパイラによってあらかじめ各段のロード命令がどの way に対応しているかを設定することで，ローカルバッファのタグ比較を除去し，ローカルバッファに way1 つ分のデータのみ保持させることによってローカルバッファの回路規模および遅延時間の最適化を行う．ローカルバッファが保持するデータが way1 つ分になると，way を選択するためのタグ比較が不要となり，way1 つが備えるローカルバッファの要素のいずれかを選択できればよい．従って，

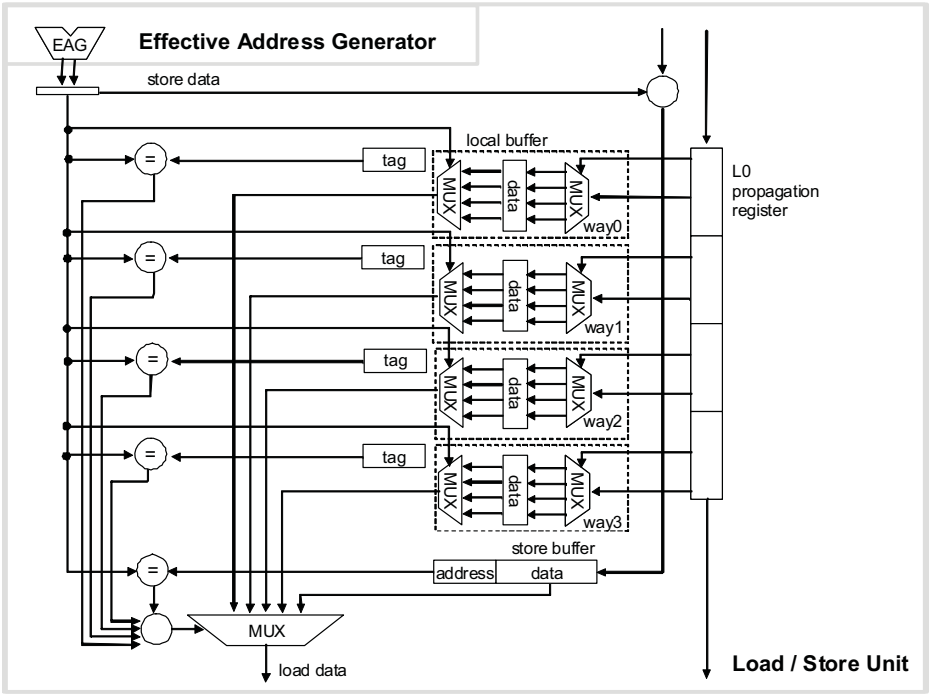

図 2 従来型ロードストアユニット

Vol.2011-ARC-196 No.18
2011/7/28

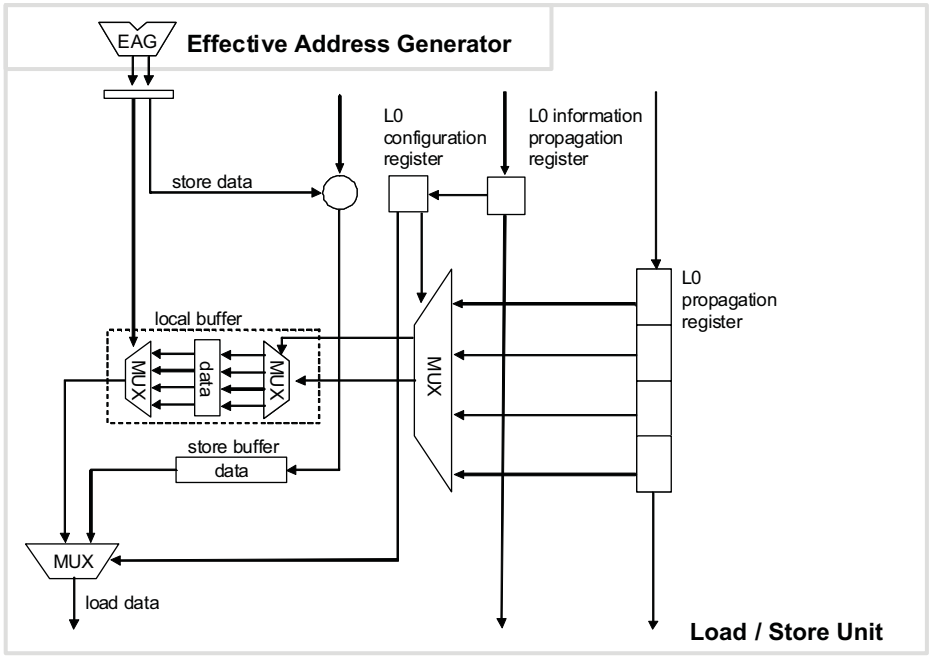

図 3 最適化したロードストアユニット

アドレス計算は下位数ビットのみの計算で済むため，EAG が簡素化され，アドレス計算とロード/ストア命令の実行がレイテンシ 1 で実行可能となる．ロードストアのレイテンシを 2 から 1 に短縮することで，ループカーネル内でロードストアのレイテンシを考慮したアレイ実行向けのスケジューリングが不要となる．

3. 最適化したロードストアユニット

ロードストアユニットでは図 2 に示す通り，ロード命令実行時はローカルバッファとストアバッファからデータが読み出され，ストア命令実行時には後段のストアバッファのデータと自段のストアデータがマージされ，ストアバッファへとデータが書き込まれる． ロード結果は後段での演算器による演算や，アドレス計算に使用されるため，後段へと伝播される．ストアバッファのデータは後段でのストア命令によるストアデータとのマージや，ロード命令による読み出しに使用されるため，後段へと伝播される．L0 伝播レジスタの値は後段の L0$に L1$からのデータを供給するために後段へと伝播される．

最適化されたロードストアユニットの回路図を**図 3** に示す．従来型ロードストアユニットの回路図を示した図 2 と比較すると way 数が 4 から 1 に削減され，タグ比較によるデータの読み出し論理が削減されている．一方，追加情報を記憶するレジスタが必要である．まず，ローカルバッファがどの way のデータを保持すればよいか設定するために，L0 情報レジスタを用意する．way 数に 2 のべき乗数の way を用いた場合，この L0 情報レジスタは各段がどの way と対応しているかを判断するために log_2（way 数）ビット，ロード対象をローカルバッファかストアバッファか選択するために 1 ビット必要とする．よって，必要な合計ビット数は段数 ×（1+ log_2（way 数））となる．さらに，各段には L0 情報レジスタからのデータを設定する L0 設定レジスタと，L0 情報レジスタのデータを後段へと伝搬させる L0 情報伝搬レジスタが必要である．L0 設定レジスタと L0 情報伝搬レジスタは，どの way に対応しているかを示す log_2（way 数）ビット，ロード対象をローカルバッファかストアバッファかを示す 1 ビットの計 1+ log_2（way 数）ビットから成る．

情報レジスタを設定するために，L0 セット命令を導入する．既存機械語命令の下位 16 ビット値セット命令（SETLO）を流用し，デスティネーションレジスタにゼロレジスタを指定する．初段のみでの通常実行ではゼロレジスタへの操作であるため，プログラムの実行にまったく影響することはなく，バイナリ互換性を維持することができる．

最適化手法によってロードストアのための EAG におけるアドレス計算を下位数ビットに簡素化した場合，ロード命令の実行に関しては問題ないが，ストア命令の実行によってストアバッファに書き込まれたデータは，アレイ最終段から L1$へと書き戻されるため，32 ビットのストアアドレスが必要となる．ここで，LAPP のアレイ実行によって L1$へと書き戻されるデータのベースアドレスは DCPL 命令の実行によって既知であるという点に注目する．DCPL 命令の詳細については文献 8) を参照されたい．さらに，ベースアドレスからアドレスの正方向または負方向に 1 ワードずつ順に書き戻されるという点に注目すると，DCPL 命令実行時に書き込み先ベースアドレスとアドレスの変化する方向を保存しておき，アレイ最終段からストアデータが L1$に書き込まれる度に 1 ワード分ずつ書き込み先アドレスを正方向または負方向にずらすことによって，アレイ実行中のストアアドレス計算は不要となる．

Vol.2011-ARC-196 No.18
2011/7/28

4. 評　　価

本章では従来型ロードストアユニット，最適化されたロードストアユニットそれぞれの回路規模と回路遅延を評価する．最適化手法の回路規模削減効果を調査するために，従来型ロードストアユニットと最適化されたロードストアユニットとの回路規模の比較を行う．アレイ段 1 段分においても回路規模の比較を行い，アレイ 1 段分での回路規模削減効果も調査する．また，最適化されたロードストアユニットにおいてローカルバッファの要素数増加により確実にクリティカルパスにならないローカルバッファの要素数を調査するために，最適化されたロードストアユニットにおいてローカルバッファの要素を変化させたときの回路遅延と加減，シフト演算を行う演算器との回路遅延の比較を行う．HDL 記述を Design Compiler D-2010.03 と 180nm テクノロジライブラリを用いて論理合成し，回路規模は 2NAND ゲート換算，回路遅延は FO4 換算にて評価した．対象モジュールのタイミング制約を速度優先に設定し最小遅延時間を求めた．

4.1 回路規模比較

本節では，従来型ロードストアユニットと最適化されたロードストアユニットそれぞれにおける回路規模の比較を単体の場合とアレイ 1 段分の場合で行う．単体の比較ではロードストアユニットの回路削減効果を調査するため，EAG の回路規模は評価に含まないものとする．**図 4** に従来型ロードストアユニットに最適化手法を適用し，ローカルバッファの要素数を変化させたときの回路規模削減率を示す．横軸はローカルバッファの要素数を示し，右側の縦軸は回路規模削減率を示す．単体比較における回路規模削減率は図 4 中の凡例で LSU を示している折れ線グラフで示す．また，図 4 に従来型ロードストアユニットと最適化されたロードストアユニットをアレイ 1 段分で評価したときの回路規模および回路規模削減率を示す．横軸はローカルバッファの要素数を示し，左側の縦軸は 2NAND ゲート換算での回路規模を示し，右側の縦軸は回路規模削減率を示す．アレイ 1 段における回路規模削減率は図 4 中の凡例で Array Stage を示している折れ線グラフで示す．図 4 中の棒グラフの凡例について LSU はロードストアユニット，EAG はアドレス計算機，EXEC は演算器，SEL はセレクタ，MAP は命令マッパをそれぞれ示す．ローカルバッファの要素数を変化させても従来型ロードストアユニットに関してはロードストアユニット以外の回路規模は変化しないが，最適化されたロードストアユニットではアドレス計算が簡素化されるため，ロードストアユニットと EAG の回路規模が変化する．

図 4 よりローカルバッファの要素数が増加すると従来型ロードストアユニットと最適化されたロードストアユニットの回路規模の差が大きくなっていることがわかる．ローカルバッファ単体で評価すると，最適化を行うことで，要素数が 4 のとき従来型ロードストアユニットから 57%の回路規模が削減でき，要素数が 128 のとき従来型ロードストアユニットから 78%の回路規模が削減できた．要素数が少ないときに削減率が少ないのは要素数に関わらず必要となるストアバッファ等が含まれているためである．従って，最適化手法により従来型ロードストアユニットの回路規模を少なくとも 57%削減可能であることが判明した．一方，アレイ 1 段で評価すると，最適化を行うことで，要素数が 4 のとき 7%の回路規模が削減でき，要素数が 128 のとき 43%の回路規模が削減できた．従って，最適化手法によりアレイ 1 段の回路規模を少なくとも 7%削減可能であることが判明した．

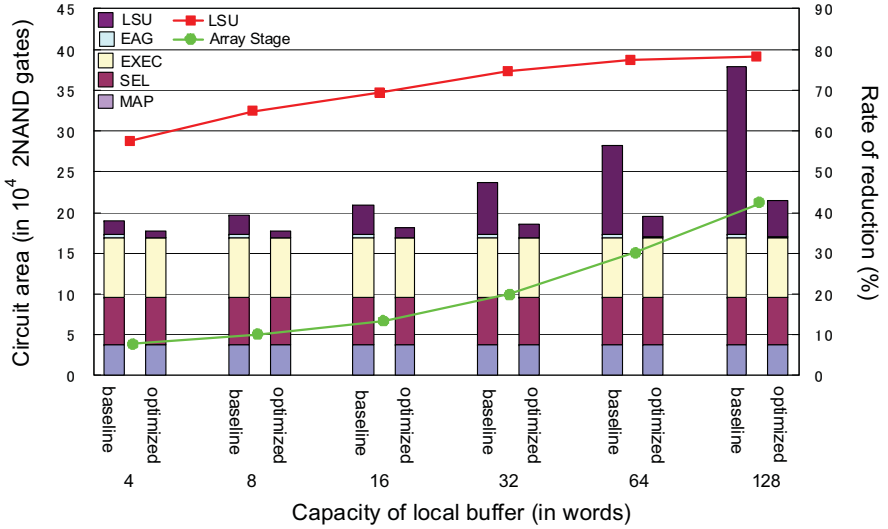

図 4　回路規模および回路規模削減率

4.2 回路遅延比較

本節では，加減，シフト演算を行う演算器と最適化されたロードストアユニットそれぞれの回路遅延の比較を行う．最適化手法において EAG でのアドレス計算は log_2（ローカルバッファの要素数）ビットの演算で行われる．**図 5** に最適化されたローカルバッファにおいて要素数を変化させたときの回路遅延と，加減，シフト演算を行う演算器の回路遅延を示

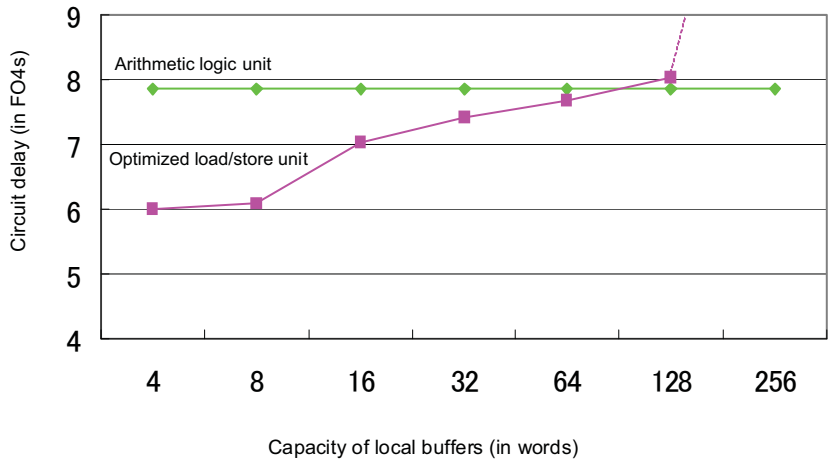

図 5 演算器および最適化したロードストアユニットにおける回路遅延

す．横軸はローカルバッファの要素数を示し，縦軸は FO4 ゲート換算での回路遅延を示す．

図 5 より，最適化されたロードストアユニットにおいてローカルバッファの要素が 64 ワードまでは加減，シフト演算を行う演算器よりも回路遅延が少ないことがわかる．従って，最適化されたロードストアユニットにおいてローカルバッファの要素が 64 ワードまではロードレイテンシを確実に 1 に削減できることが判明した．ローカルバッファの要素数が 8 ワードまでは回路遅延が増加しないが，これはローカルバッファの要素数が少ないためストア命令実行によるストアバッファへのデータ書き込みのパスが，ローカルバッファおよびストアバッファから値を読み出すパスよりも長くなるためである．また，これまで評価した画像処理プログラムでは従来のローカルバッファにおいて 1way あたり 16 ワードの要素数で十分であることがわかっているため，ロードレイテンシ削減効果は確実にあることも判明した．

5. むすび

本稿では LAPP のアレイ実行中におけるロード命令と way 番号が 1 対 1 に対応していることに注目し，事前にロード命令に対応する情報を既存機械語命令の実行により追加することによって，ローカルバッファの way およびタグ比較回路を削減し，ロードストアレイテンシも削減する手法を提案した．最適化手法を用いると，ローカルバッファの要素が 64 ワードまでは，ロードのレイテンシを 2 から 1 へ確実に削減可能であることが判明した．

本稿では L0 情報レジスタに値を設定する L0 セット命令は手動で追加したが，今後はコンパイラまたは命令トランスレータによって自動的に L0 セット命令を追加することを予定している．また，これまで評価した画像処理プログラムにおいては 16 ワードより多い要素を持つローカルバッファは不要であるが，さらに多くの要素数を持つローカルバッファを必要とするアプリケーションが存在するか調査が必要である．

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社の協力で行われたものである．

参考文献

1) Seiler, L., Carmean, D., Sprangle, E., et al.: Larrabee: A Many-Core x86 Architecture for Visual Computiing, *IEEE Micro*, Vol.29, No.1, pp.10–21 (2005).
2) Bouwens, F.J., et al.: Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array, *HiPEAC'08*, pp.66–81 (2008).
3) 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).
4) Hyunchul, P., Yongjun, P. and Scott, M.: Polymorphic pipeline array: a flexible multi core accelerator with virtualized execution for mobile multimedia applications, *MICRO 42*, pp.370–380 (2009).
5) 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).
6) Kazuhiro, Y., Takuya, I., Takashi, N., Jun, Y., Hajime, S. and Yasuhiko, N.: An Instruction Mapping Scheme for FU Array Accelerator. IEICE Transactions on Information and Systems, Vol. 94-D, No. 2, pp.286–297 (2011).
7) Yong, C., Huaiyu, Z. and Xian-He, S.: An Adaptive Data Prefetcher for High-Performance Processors, *CCGRID*, pp.155–164, (2010)
8) 森　浩大，岩上拓矢，吉村和浩，中田　尚，中島康彦：演算器アレイ型アクセラレータのための命令変換手法の検討，2010 年並列／分散／協調処理に関する『金沢』サマー・ワークショップ (SWoPP 金沢 2010), 情報処理学会研究報告，Vol. 2010-ARC-190, No. 26, pp.1–6 (2010).