Vol.2013-ARC-206 No.14
2013/7/31

リング型アレイアクセラレータのマクロパイプライン化による性能見積もり

藤原 知広[†1]　姚 駿[†1]　原 祐子[†1]　中島 康彦[†1]

概要：我々は，近年注目されている大規模シミュレーションやビッグデータといった並列度の高いプログラムの実行速度を高めるリング型アレイアクセラレータとして EMAX(Energy-aware Multimode Accelerator eXtension) を提案している．EMAX はプログラム内の命令を演算器アレイユニットにそれぞれ命令を割り当てることで，命令を一括処理し，並列度の高い 2 重ループの最内ループ内のプログラム等を高速に実行し，並列度の低い部分はホストの動作周波数の高い既存のプロセッサを用いて実行する．しかし，このアクセラレータの演算処理速度に比べ，ホスト・アクセラレータ間の DMA 転送によるデータ通信速度が比較的遅く，アクセラレータの演算処理をデータ通信とオーバーラップさせる必要がある．1 ポートメモリを用いて効率よくプリフェッチを行う仕組みを導入することで，演算処理とデータ通信の同時実行を行う．これにより，処理前のデータの転送とホストへの処理後のデータの保存を同時に行うことが可能となるため，シミュレータを用いて評価を行った結果，EMAX 全体の実行時間が 27% 減少することを確認した。

1. はじめに

現在，大規模なシミュレーションを行う科学技術計算や画像処理といった，処理対象となるデータ量が多く，且つ並列性の高い計算を求められる機会が年々増加している．このような計算をより高速に実行するため，GPGPU 等の外部処理装置の組み合わせにより演算処理能力向上が図られている．しかし，このような構成では，マイクロプロセッサとアクセラレータ間のデータ転送速度は，DMA 転送を用いて高速にデータ転送を行っているにもかかわらず，ボトルネックとなる場合が多く，データ転送を隠蔽する必要がある．

我々は，演算処理の高速化と省電力化を目的として，リング型アレイアクセラレータ (Energy-aware Multimode Accelerator eXtension : EMAX)[2] を提案している．シングルポートメモリと演算器を組み合わせたユニットを 2 次元的に並べた構造を持ち，ある程度の大きさのデータを一度にローカルメモリに格納し，細いデータパスにおけるデータのやり取りを削減することで，ボトルネックとなるデータ転送回数を減らし，高速化を図っている．実行を最適化するため，ホストからのデータ転送とアクセラレータの演算をオーバーラップさせ，パイプライン処理を行うモデルを構築し，シミュレータにより評価した．

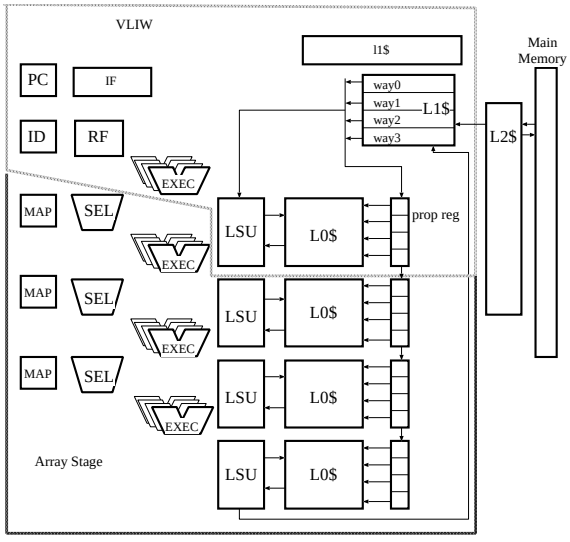

図 1　LAPP の構成概要

本稿では，2 章において，先行研究である画像処理向け線形アレイ型アクセラレータ (LAPP:Linear Array Pipeline Processor)[1] の構成と LAPP の課題について述べる．3 章において，LAPP の課題を解決するために，我々が提案する EMAX のシステム構成について述べ，4 章にてマクロパイプライニング手法の概要と効果について述べる．そして，5 章にてシミュレータにより行った性能評価結果を示す．

[†1] 現在，奈良先端科学技術大学院大学
Presently with Nara Institute of Science and Technology

2. 先行研究

我々は EMAX 以前に，画像処理や格子法による計算で用いられる，対象となるデータの隣接要素を参照しながら，それぞれの要素を独立に演算を進めるステンシル計算に強い線形アレイ型アクセラレータ （Linear Array Pipeline Processor : LAPP）を提案していた．本章では，LAPP の概要と課題について述べる．図 1 に，アレイ段数が 4 段の LAPP の概要を示す．LAPP は従来の VLIW プロセッサを線形に拡張した構造をしており，プログラムの中に含まれるループ処理の命令列を演算器アレイに写像することにより高いスループットで演算を行う．また，バイナリ互換があるため既存の機械語命令列を扱え，専用のコンパイラが必要なく，過去のソフトウェアを再利用できる点が特徴である．LAPP は従来型の VLIW 型プロセッサである初段と，後段のアレイステージで構成される．アレイ段数は必要に応じて設計時に決定する．LAPP 全体は，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$），命令キャッシュ（I1$）で構成される．アレイ部は従来型 VLIW プロセッサの演算器を線形に接続した構造を持ち，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）で構成される．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 (prop reg) を備える．

LAPP は，演算器にロードするデータのアドレスが単調に変化する処理（画像処理等）に適した構造を持っていたが，(1) 下段にロードするデータを伝搬するためにはレジスタを経由しなければならないため，演算で用いる配列数が増加するほど下段に伝搬すべきデータバスも増加することや，(2) 狭い範囲のランダムアクセスにしか対応できない点，(3) ストア可能な way が少ないので，ストアする配列数が多い演算には適していないことや，(4)L1 自体の way の少なさによりロード可能な配列数が限られてしまう課題がある．

3. EMAX の概要

本章では，先行研究における課題の解決方法を示し，解決方法を元に改良した EMAX 全体の構成と動作の説明を行ったあと，マクロパイプライニングの概要と効果について説明する．

3.1 LAPP の課題解決策

LAPP の課題 (1)，(2) を解決するためには L1 データキャッシュから L0 データキャッシュにデータをロードする構造を変え，各 EAG に対してキャッシュを分散配置し，それぞれが独立にアドレスを指定してデータをロードできる構造が必要となる．課題 (3) は，分散配置した全てのキャッシュに対しストア命令を用いて演算結果をキャッシュに格納できる構造にすることで解決できる．課題 (4) は全体としての再利用可能なデータを増やすことで可能となる．このような方法で LAPP の課題を克服しつつ，ステンシル計算に強みを持つ LAPP の特徴を受け継いだアクセラレータとなる EMAX の具体的な構成を決定した．

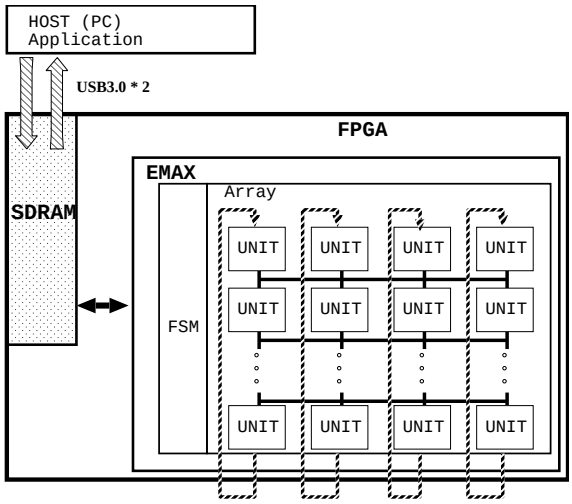

図 2 アクセラレータの全体構造

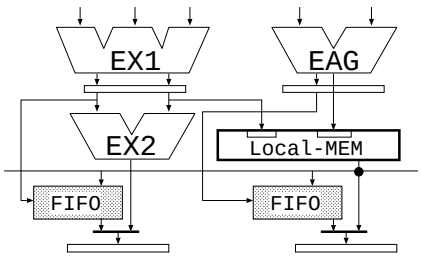

図 3 ユニットの構造

3.2 EMAX の構造

EMAX 全体の構成を図 2 に示し，EMAX 内のアレイユニットの構造について図 3 に示す。LAPP には存在していた汎用プロセッサ部分を取り払い、ホストの持つ高い動作周波数のプロセッサでプログラムを実行し，2 重ループの最内ループ等並列化可能な部分を処理する場合に，EMAX を呼び出して実行する．ホストがソフトウェアによるスケジューリングを行い，制御情報（各演算器への命令，ユニット間のセレクタとバスの設定）と演算データを SDRAM に転送する．制御情報を各ユニットに転送し，ローカルメモリへのロードが必要なユニットには演算データをローカルメモリにプリフェッチする．その後，演算を行い，演算結果をストアしたローカルメモリからデータを取り出して SDRAM に戻し，ホストが演算結果を得る．EMAX の 1

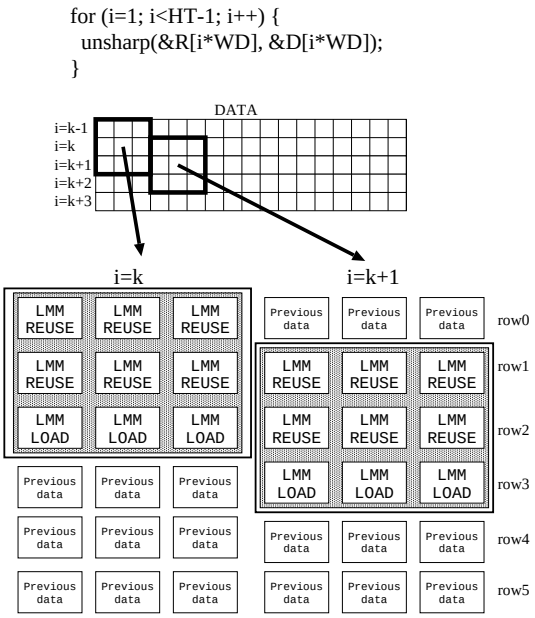

図 4 リング構造のデータ再利用用例

つのユニットは，数値計算用の EX1，EX2 が各 1 個，アドレス計算用の EAG 1 個の計 3 個の演算器及び，ローカルメモリ 1 個を加えた構成である．このユニットを 2 次元配置する．また，EAG の他に EX1 も EAG として用いることで，1 ユニットあたり最大 2 つのロード/ストア命令を記述可能にした．これらにより，LAPP の問題点であった 1 ループ内で実行可能なロード/ストア命令数が少ないという点を改良した．

再利用できるデータを増やすために，EMAX はリング構造を持つ．図 4 に 3 × 3 のステンシル計算におけるローカルメモリへのデータ読み込み例をもとに，リング構造を利用したデータの再利用例を示す．EMAX 命令は，i 回目のループ実行が終わると，$i+1$ 回目のループ実行の命令マッピング開始位置を，i 回目のマッピング開始位置からのシフト量で指定する．ステンシル計算の i 回目と $i+1$ 回目の実行において，row 1 と row 2 のローカルメモリにプリフェッチするデータは共通なので，SDRAM からデータを読み出さず，命令マップをシフトによりデータを再利用し，ロールメモリのデータ更新に伴う電力や時間を削減できる．

3.3 各ステージ概要

EMAX の動作は大きく図 5 に示す 5 つのステージからなる．

3.3.1 Stage1（HOST から SDRAM）

EMAX に必要な制御情報と演算データをシステムコールを用いて HOST から SDRAM へ転送する．その時に，2 つの USB インタフェースの内の 1 つを HOST からの転送専用としてアクセラレーションに必要なデータを転送し続ける．また，同じデータを連続で転送しないために，転送情報を保持して次の実行時の転送データと比較し，一致した場合はシステムコールを行わない．パイプライン処理を行うためには，1 ステージ前に SDRAM に転送された制御情報を用いる必要がある．制御情報を保存する空間を 2 箇所設けて常に SDRAM への読み出しと書き込みが交互に行われる形にすることで，ホストの書き込みとローカルメモリの読み出しが同じ空間に同時に行われることを防ぎ，SDRAM 自体にパイプラインレジスタの役割をもたせた．

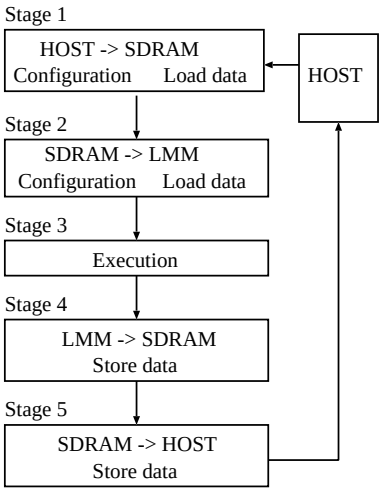

図 5 各ステージの概要

3.3.2 Stage2（SDRAM からローカルメモリ）

SDRAM から受け取った制御情報に基づき，各ユニットのバスの構成を決定する．この際，ループの回数が図 8 の $i > 0$ にあたる場合には，命令マップの更新は行わずに，制御情報内で指定された段数分だけ命令マップをシフトする．その後，各ユニットのレジスタを初期化し，ユニットごとの制御情報をもとにローカルメモリのタグ情報の初期化とロードを行い．DDR3 からローカルメモリへのデータの読み込みも更新する必要がない場合は，既にローカルメモリに読み込み済のデータを再利用して演算を行う．マクロパイプライン化する場合には，新たにプリフェッチ命令を用意し，各ステージを同時実行できるようにする．

3.3.3 Stage3（演算）

演算を開始する．マクロパイプライン化した場合には，ユニットは演算処理以外にもデータ保存と転送を行い，3 つの作業を並列処理することで，転送時間を隠蔽する．

3.3.4 Stage4（ローカルメモリから SDRAM）

EMAX の制御情報を元に該当するローカルメモリから SDRAM に演算結果を読み出す．マクロパイプライン化した場合には，ローカルメモリからのストアデータのアドレスの読み込みと，実際にローカルメモリへストアする命令と区別する必要があるため，マクロパイプライン化を実現するにあたって新たに用意する．両者の命令マッピングのシフト量に合わせて配置することで，並列処理が可能となる．

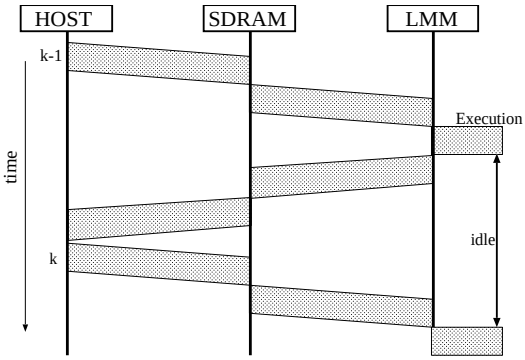

図 6 EMAX の動作（マクロパイプラインなし）

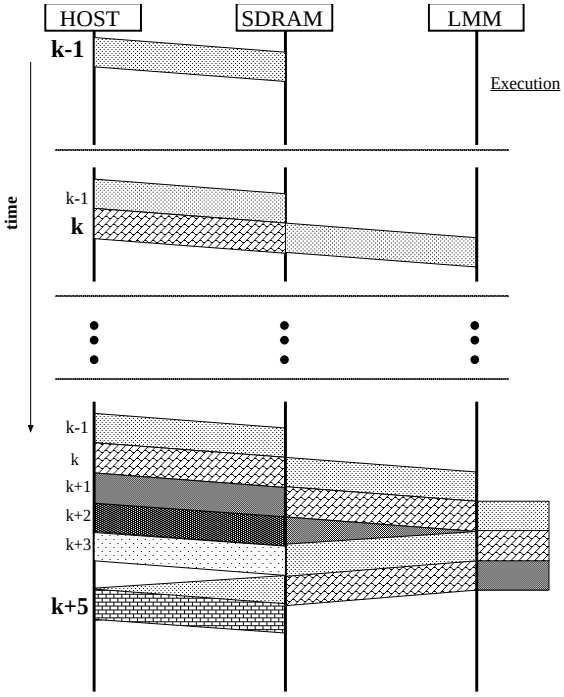

図 7 EMAX 動作 (マクロパイプライン有り)

3.3.5 Stage5（SDRAM から HOST）

SDRAM に格納された EMAX の演算結果をシステムコールを用いて SDRAM から HOST へ転送する．その際には，Stage 1 で使用していない方の USB を用いてデータ転送を行う．

EMAX のデータと実行流れを図 6 に示す．この図は下に行くほど時間が経過したことを表しており，HOST，SDRAM，LMM（ローカルメモリ）の間の矩形の左右の辺の長さが処理時間を表す．この図の中の $k-1$ 回目のアクセラレータ実行に注目すると，マクロパイプライン化されていない場合はホスト－ SDRAM 間や SDRAM － LMM 間のデータ転送時にはアクセラレータの演算が行われておらず，長い待機時間が存在する．この待機時間短縮し，常に演算前のデータ転送をしつつ，同時に演算処理後のデータを受け取る仕組みに構築し，HOST-SDRAM 間の細いデータバスを最大限に利用することができる．

```
for(i=0;i<240;i++){
  #prefetch[i+2][]
  for(j=0;j<320;j++){
    a[i][j] = b[i-1][j-1]+b[i-1][j]+b[i-1][j+1]+
              b[ i ][j-1]  +b[ i ][j]  +b[ i ][j+1]+
              b[i+1][j-1]+b[i+1][j]+b[i+1][j+1];
  }
}
```

図 8 プリフェッチサンプルコード

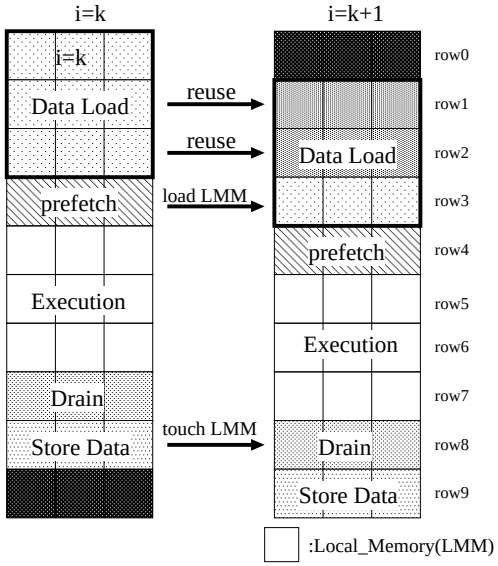

図 9 マクロパイプライン時のローカルメモリの動作

4. マクロパイプライン

マクロパイプラン化した EMAX の動作を図 7 に示す．CPU の命令パイプラインのように命令実行をいくつかのステージによって区切るのでなく，ホストと SDRAM，SDRAM とローカルメモリの間のデータの転送と演算をオーバーラップさせた．演算器の待機時間とデータの転送をオーバーラップすることにより，データ転送時間を隠蔽して，アクセラレータが常に演算処理を行っている状態を維持することによって高速化を図る．EMAX は，各々独立したローカルメモリを読み出し専用，書き込み専用として動作させることによりマルチポートメモリを必要としない．本章では，5 ステージのマクロパイプラインを実現するための，各ステージの動作概要について述べ，シングルポートのメモリのみでパイプライン処理を実現するモデルについて記述する．

4.1 シングルポートメモリによるパイプライン動作

データ転送と演算のパイプライン処理を行うには，k 回目の演算に必要なデータを $k-1$ 回目までにローカルメモリに格納する必要がある．このためにプリフェッチ命令が必要となり，ローカルメモリはデータのロード（Stage2)

表 1　使用する評価プログラム

フィルタ名	演算命令数	ロード命令数	ストア命令数	初回ロード [byte]	定常状態ロード [byte]	ストア [byte]
tone_curve	1	5	5	2048(4096)	2048	1280
hokan1	19	12	12	3840(5120)	1280	1280
hokan2	11	11	11	2560(5120)	2560	1280
expand4k	43	10	10	3840(5120)	1280	4096
unsharp	22	10	10	3840(5120)	1280	1280
blur	39	16	16	3840(5120)	1280	1280
edge	9	8	8	3840(5120)	1280	320

( ) 内はマクロパイプラインの場合

と同時にストア命令による演算後データのローカルメモリへの格納（Stage3) と SDRAM へのドレイン（Stage4）という 3 つの処理を同時に行う必要がある．しかし，マルチポートメモリを導入すると面積効率が著しく低下する．代わりに，3.2 節で述べたリング構造と各ユニットへのローカルメモリの分散配置がこの問題を解決する．図 8 にプリフェッチ命令を含んだ EMAX が行う 3 × 3 のステンシル計算のサンプルコードを示し，図 9 では，図 8 のコード実行時のリング構造による命令マッピングのシフト量が row1 段分を仮定した場合のローカルメモリの役割分担について示す．また，図 9 はループ実行の定常状態時のローカルメモリの様子を 2 連続分取り出し，図中の格子 1 マスはユニットのローカルメモリを表している．EMAX は図 8 の外側ループの演算が一回実行するごとに命令マッピングを下段へシフトしながら演算を行う．命令マッピング時に一定の間隔でローカルメモリに次の演算に必要なデータをロードすることで，ローカルメモリへのプリフェッチをすることがができる．また，演算後データは，ローカルメモリからのデータロードや，先程のようなプリフェッチ対象以外のユニットのローカルメモリへストアし，ローカルメモリから SDRAM へデータのドレインも命令シフト量を意識して行うことにより，3 つの処理を同時に行う．ユニットごとに分散配置したシングルポートメモリがそれぞれ別々に動作することにより，マルチポートメモリの機能を代替している．

5. 性能評価

本章では，マクロパイプライン化の評価を行う．評価プログラムのサンプルを示したあと，評価プログラムの命令種類の割合を示し，評価プログラム別実行速度，各ステージ別実行速度，理想的なインターフェースを仮定した実行速度について評価を行う．

5.1 評価プログラム

評価プログラムの一覧とその特徴を，表 1 に示す．評価プログラムは色調補正 (tone_curve)，SAD 計算 (hokan1)，中間値補正 (hokan2)，鮮鋭化 (unsharp)，拡大補間 (expand4k)，メディアンフィルタ (blur) を使用した．表中には、評価プログラム中の演算命令数、演算結果のストア命令数、ローカルメモリからユニットへのロード命令数、SDRAM からローカルメモリへの初回ロード時のデータサイズ、定常状態 (k 回目の実行) で 3.2 節で記述したような処理対象のデータが再利用が可能になった場合の SDRAM からローカルメモリへのロードするデータサイズ、ローカルメモリから SDRAM への転送データサイズを示している。評価プログラムにはそれぞれ特徴があるため、EMAX の実行速度にどのように影響するか評価を行った．

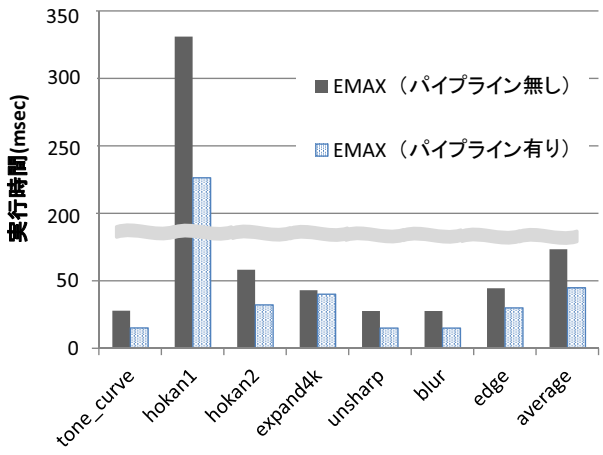

図 10　マクロパイプラインの有無による実行速度比較

5.2 評価結果

EMAX のクロックレベルシミュレータにより評価を行った．想定した EMAX のハードウェア性能は，ホストとの通信速度には，USB3.0 を仮定し、EMAX を搭載した FPGA 上に実装することを想定して評価を行った。また，USB3.0 自体は FPGA に対して 2 本の接続を仮定した。マクロパイプライン動作させる場合には、それぞれをリード専用，ライト専用のデータバスとして使用する。転送速度は実測値を元にリードが 50Mbps、ライトが 25Mbps とし、EMAX の動作周波数は 50Mbps とする．SDRAM は 1 GB の容量を持つ DDR3 メモリ、ホストには Intel の Core i7-X3970 PC を用いている．

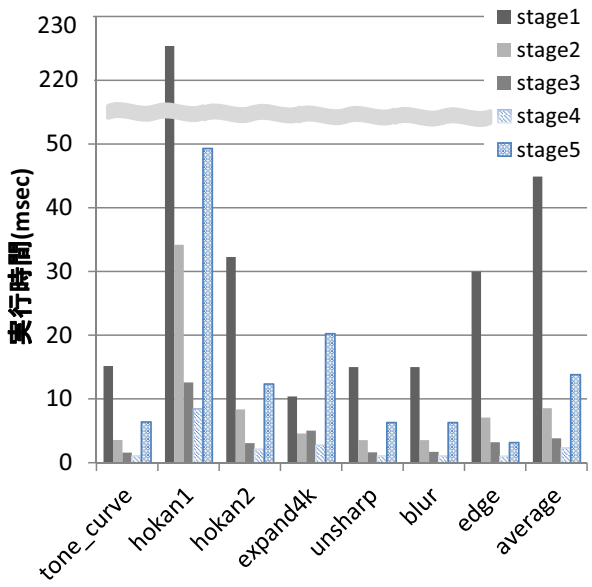

図 11　評価プログラムごとの各 Stage における実行速度比較

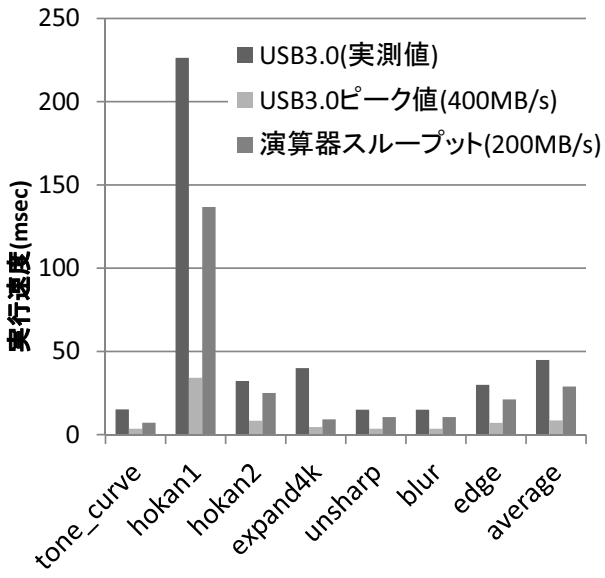

図 12　マクロパイプライニングにおける理想的なインターフェース仮定した実行速度比較

5.2.1　マクロパイプラインの有無による実行速度比較

評価プログラム別の測定結果を図 10 に示す。平均約 27% の実行時間の短縮ができることが分かった．しかし，5 ステージのパイプライン処理を行っているにもかかわらず，処理時間の短縮が殆ど見られない評価プログラムも存在した．これは，USB3.0 のデータ転送がアクセラレータの演算時間よりもはるかに大きいボトルネックになるからであり，プログラム実行時間全体における HOST-SDRAM 間の転送時間の割合が大きいほど効果が薄い結果となった．

5.2.2　各 Stage ごとの実行速度比較

各マクロパイプラインのステージごとの処理時間を分析した結果を図 12 に示す．マクロパイプライン化の効果が少なかった評価プログラムは，USB3.0 の読み込みと書き込みの実行時間の差が大きく，パイプラインのバランスが悪いという結果が得られた．そして，マクロパイプライニングしない場合はプログラム実行時間全体の約 80% が USB3.0 の送受信に費やされる．USB3.0 を用いないステージのみに注目すると、LMM-SDRAM 間の転送の時間が演算時間よりも長く，特に演算結果のストアデータをローカルメモリから SDRAM に転送する際に時間がかかっていることが分かった．

5.2.3　理想的なインターフェース仮定した実行速度比較

プログラムの実行速度において，5.2.1 節の評価でボトルネックとなっていた USB3.0 の実測時間を用いず，他のインターフェースで HOST と EMAX を接続した場合を仮定し，HOST-SDRAM 間のデータ転送を EMAX の演算スループットと同一のスループットを持つインターフェースと USB3.0 のピーク値でデータ転送を行ったと仮定した場合の見積もりを図 12 に示す．EMAX の実行時間は USB3.0 の実測値でデータ転送に比べ，約 81% ，演算器スループットと同一のスループットを持つインターフェースでのデータ転送仮定した場合は，約 36% の実行時間の短縮をすることができる予想となった．

6.　むすび

本稿では，従来の EMAX がホストからのデータ転送中は演算が行われないことに着目して，新規命令の追加とマクロパイプライン化を用いてデータの転送と演算をオーバーラップさせて動作の最適化を行った場合をシミュレータで評価し，その見積もりを行った．USB3.0 のデータ転送速度の実測値に基づいて見積もった場合，マクロパイプライン化により約 27% の性能向上が見られた．一方 USB3.0 の理想ピーク性能に基づいて見積もった場合，81% の性能向上が可能であること，また，演算性能と同一のスループットの USB を仮定した場合，36% の性能向上が可能であることが分かった．

7.　謝辞・参考文献

なお，本研究の一部は科学研究費補助金（基盤（A) 課題番号 24240005、萌芽課題番号 24650020 および若手研究（B）課題番号 23700060）および JST-ASTEP 探索タイプ（課題番号 AS242Z02732H）による．

[1] 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).

[2] 王昊, 姚駿, 中島康彦: ”多様なアクセスパターンに適応するアクセラレータ向けメモリアクセス機構”, 研究報告計算機アーキテクチャ（ARC）, Nagasaki, Mar. (2012)