Vol.2013-ARC-203 No.9
2013/2/1

GCCのvectorizerを利用した演算器アレイ向け命令変換手法

王 昊[1]　姚 駿[1]　中島　康彦[1]

概要：
多数の演算ユニットを備えるGPGPUでは，CUDA等明示的な並列処理の記述が必要なプログラミング言語を採用することにより，ハードウェアの差異を隠蔽することと，処理の高速化を両立している．ただし，所望の性能を引き出すためには，ハードウェア構造の理解と，相当のチューニングコストが必要である．一方，我々は，演算速度向上と消費電力低減の両立を目的として，演算器とローカルメモリの組を多数配置する構成の演算器アレイ型アクセラレータ（LAPP）を提案してきた．しかし，従来のLAPP[2]には，既存のVLIW命令列にプリフェッチ情報を挿入するだけで，イタレーション間に依存関係のないループを高速実行できる利点がある代わりに，適用可能なループに制約がある．また，命令セットが異なる基本プロセッサに適用するためには，アクセラレータ部分を新たに設計する必要がある．本稿では，LAPPの実行方式を踏襲しつつ従来の制約を緩和する新たなアクセラレータ構成方式，および，GCCのvectorizerを利用する命令生成方式について述べる．現在，Uncprop情報に基づき，コントロールフロー解析，データフロー解析，および，メモリアクセスパターン解析を行い，簡単な構造のループに対して，アクセラレータ用命令列を生成できる段階にある．簡単なプログラムに対して適用したところ，LAPPに比べて，平均65%の命令行数を削減できることがわかった．また，３２行構成を仮定した場合，行数の削減により生じた空き演算器を使用すると，LAPPに比べて，２倍から８倍の性能向上を期待できることがわかった．

1. 背景

これまで，プロセッサの高速化は，科学技術計算やマルチメディア処理が利用可能な計算能力を増大させ続けてきたものの，近年は，電力効率も重視されるようになってきた．現在では，処理速度向上と消費電力低減を両立する実用的な仕組みとして，GPGPUが広く使われ始めている．GPGPUの高い演算性能を引き出すためには，GPGPUのハードウェア構造に合わせて，データレベル並列性の高いプログラムを記述しなければならない．GPGPU向けコンパイラに関して様々な研究が行われている[1]ものの，依然，ハードウェアの特性に関する知識が必要であり，また，GPGPUの世代が変わると特性も変化する．

一方，我々は，演算速度向上と消費電力低減の両立を目的として，演算器とローカルメモリの組を多数配置する構成の演算器アレイ型アクセラレータを提案してきた．しかし，従来のLAPPには，既存のVLIW命令列にプリフェッチ情報を挿入するだけで，イタレーション間に依存関係のないループを高速実行できる利点がある代わりに，適用可能なループに制約がある．具体的には，各演算器に届けることが可能なメモリデータは，初段に装備されたローカルメモリから送り込まれてくる比較的小さなアドレス範囲のデータであり，また，アドレス範囲の変化が単調であることを前提としている．さらに，演算数に比べてストア命令数が少ないことも前提としているため，画像処理のように多くのデータをロードし，少ないデータをストアするループには適するものの，ランダムなメモリアクセスや，多くのストアを必要とするループには適さない．この問題を解決するためには，LAPPの初段が備えるローカルメモリを分散させ，どのメモリに対してもロードストア可能な均一な構成とする必要がある．

また，LAPPが採用する構成方式の場合，命令セットが異なる基本プロセッサに適用するためには，アクセラレータ部分を新たに設計する必要がある．この問題に対しては，基本プロセッサとアクセラレータ部分を切り離しつつ，オーバヘッドの小さい実行方式とする必要がある．

本稿では，LAPPの実行方式を踏襲しつつ従来の制約を緩和する新たなアクセラレータ構成方式，および，GCCのvectorizerを利用する命令生成方式について述べる．特に，アクセラレータのハードウェア構造に対応するために，

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

表 1　抽出したメモリ参照パターン.
Table 1　Typical memory access pattern.

名称	パターン例および動作
P1	A+x, B+y, C+z ベースアドレス A,B,C は固定．x,y,z は広範囲のランダム値
P2	A, A+a1, A+a2, A+a3, A+a4, A+a5 ベースアドレス A が単調変化．a1-a5 は固定オフセット
P3	A, A+a, A+b, A+c, A+d ベースアドレス A が単調変化．a, b, c, d は狭い範囲のランダム値
P4	A+a1, A+a2, A+a3, A+a4 ベースアドレス A が 2 単位ずつ単調変化．a1,a2,a3,a4 は固定オフセット
P5	A, A+a1+x, A+a2+y, A+a3+z ベースアドレス A が単調変化．a1,a2,a3 は固定オフセット．x,y,z は狭い範囲のランダム値

GCC の vectorizer を利用して，最内ループと多数の演算器を直接的に関連づける命令生成手法を提案する．以降，2 章では，新たに提案するアクセラレータのアーキテクチャについて説明し，3 章では，提案する命令生成手法について詳述する．4 章では，さらに，ループを複数に分割して並列実行する命令生成手法について紹介する．5 章では，提案手法について考察し，今後の課題について述べる．

2. 対応したい基本ループ構造とアクセラレータの構成

本章では，どのような基本ループへの対応を考えているか，また，対応するために必要なアクセラレータの構成について説明する．

2.1 基本ループ構造

[3] では，従来の画像処理に必要なメモリ参照パターンに加え，さらに適用範囲を広げるために必要となるメモリ参照パターンを整理し，代表的な 5 種類を抽出した（**表 1**）．P1 は，固定ベースアドレス A，B，C に，命令が生成するランダムオフセット x，y，z を各々加えたアドレスからロードする．P2 は，単調変化するベースアドレス A に，5 種類の固定オフセットを加えたアドレスからロードする．P3 は，単調変化するベースアドレス A に，4 つの狭い範囲のランダムオフセットを加えたアドレスからロードする．P4 は，毎サイクル 2 単位ずつ単調変化するベースアドレス A に，4 つの固定オフセットを加えたアドレスからロードする．P5 は，単調変化するベースアドレス A に，各々固定オフセットと狭い範囲のランダムオフセットを加えたアドレスからロードする．

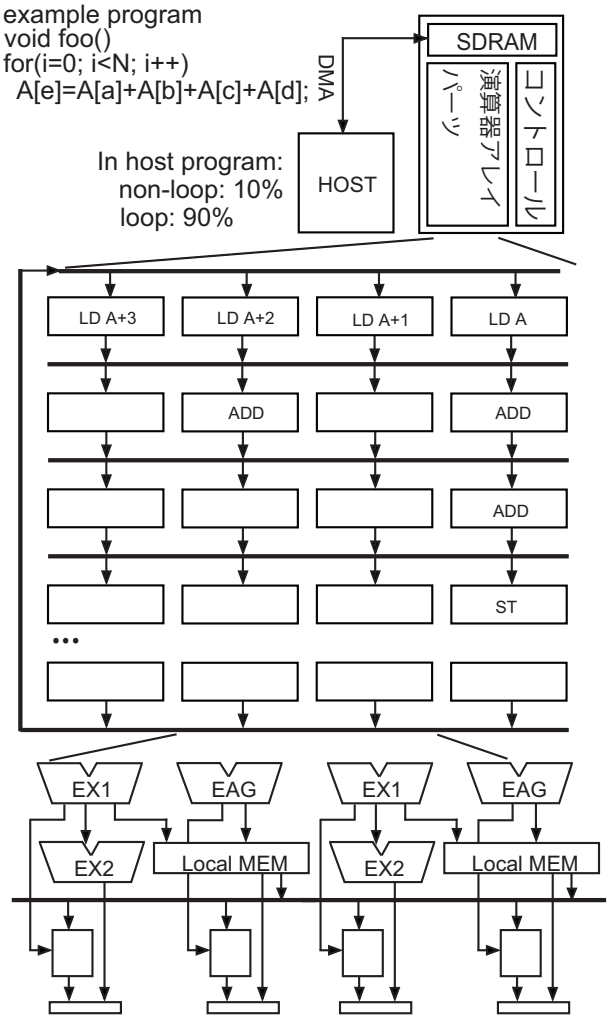


図 1　アクセラレータの全体構成.
Fig. 1　Overall of the structure of an accelerator.

2.2 アクセラレータのインタフェース

以上のようなメモリアクセスパターンを基本命令セットに依存しないアクセラレータに通知して実行するためには，コンパイラによる命令列生成，アクセラレータが必要とするレジスタ等の初期値，および，アクセラレータへの制御情報の送信手段が必要である．本提案では，まず，GCC の vectorizer を利用してアクセラレータの詳細な構成の差を意識しない中間命令列を生成し，次に，デバイスドライバがアクセラレータのハードウェアの構成に合わせて詳細な制御情報（演算種別の写像や内部セレクタの切り替え情報）を生成し，処理対象となるデータとともに，アクセラレータに送り込む方式とした．この際，制御情報の生成を高速化するために，過去に生成した情報を蓄積して再利用することを想定している．

2.3 アクセラレータの構成

図 1 に示すように，提案するアクセラレータは，演算器とシングルポートローカルメモリを二次元配置している．

従来の LAPP と異なり，アクセラレーション不可能な命令列を実行する基本プロセッサ部分は備えていない．HOST が，各演算器の演算種別，セレクタ設定情報，バス切り替え情報を生成し，処理対象データとともに制御情報としてアクセラレータに転送する．このために，HOST とアクセラレータの間に SDRAM を配置しており，アクセラレータ動作中にも次のデータを送り込んだり，演算結果を取り出すことを可能としている．各行は４つのユニットから構成されると仮定し，各ユニットは，カスケード接続された２つの演算器（EX1 および EX2）とローカルメモリ（アドレス生成用の EAG を含む）を１つ，また，ローカルメモリから読み出したデータをバス経由で一時的に保持する FIFO を備えている．これらを３２行分配置し，さらに上端と下端を接続し，全体としてリング構造を構成している．

EX1 と EX2 には，独立した演算を写像したり，２つを連結して浮動小数演算を写像することを想定している．ローカルメモリから読み出したデータは，そのままユニット下端のレジスタに格納し，次のサイクルで次段の演算器に投入したり，バスを経由して，隣接するユニットの FIFO に格納した後に，FIFO の容量の範囲内で前後のデータを参照できる構成としている．この仕組みが，狭い範囲のランダム参照を可能とするための鍵となっている．各ユニットの EX1 と EAG に，それぞれに FIFO を装備していることから，１行あたりに実行可能なロード命令は８個である．このうち４つが EX1 と FIFO で実現可能な狭い範囲のオフセット参照に対応し，残り４つは EAG とローカルメモリで実現可能な広い範囲のオフセット参照に対応する．

図１に示すプログラムでは，配列 A に N 個の要素が入っている．毎サイクル，単調増加するベースアドレス A に４つの異なる固定オフセットを加え，４つのロードを実行する．ロード結果を加算して，最終的に別のユニットのローカルメモリにストアする．この実行モデルは LAPP と同様であるものの，ロードに使用するメモリとストアに使用するメモリを分離できるため，全てをシングルポートメモリにより構成することができる．

従来型のパイプラインプロセッサでは，マルチポートメモリを使用しない限り，４つのロードには，最低４サイクルを必要とし，演算を並列実行できると仮定してもループの１イタレーションを実行するには，最低５サイクルを要する．N 回実行するには，5N サイクルを要する．しかし，本アクセラレータでは，４サイクル分のデータをローカルメモリから FIFO にプリフェッチしておくことにより，次のサイクルから，毎サイクル，ローカルメモリから１要素をロードしつつ各 FIFO に供給し，各 FIFO から合計４つのデータを同時にロードできる．このため，4+N サイクルで済む計算になる．

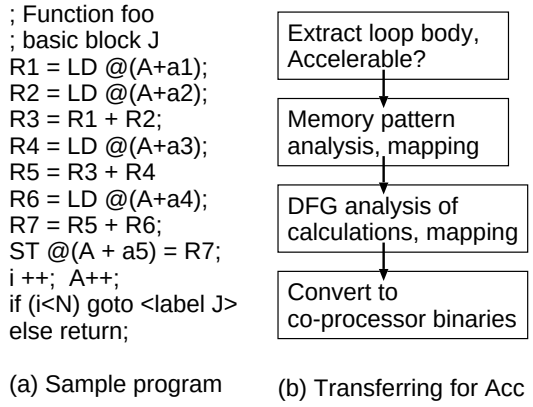


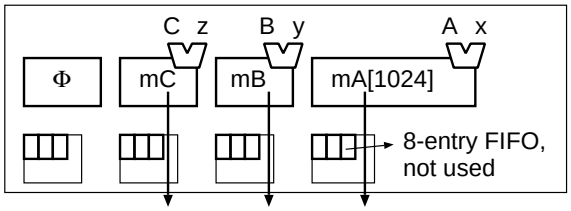
図 2　命令生成の流れ．
Fig. 2　Flow of the generation of binary for accelerator.

3. 命令列生成手法

本章では，提案するアクセラレータの構成に対応できる命令列生成手法を述べる．図 **2** に，命令生成のフレームワークを示す．まず，GCC が生成する，機種やプログラミング言語に依存しない中間表現（(a) に対応する情報）を利用する．この表現は，GCC がソースプログラムの関数単位に処理を行い，基本ブロックに関する解析結果として，メモリアクセスパターン，演算種別，命令また関数間の依存関係などの情報を含んでいる．本提案では，これらの情報を使用しながら，関数単位に処理を行い，アクセラレータにより高速実行可能なループを選択する．画像処理の場合，ループ回数は固定であることが多い．提案するアクセラレータのハードウェア構成は，ループの回転数をあらかじめ指定しておくことにより簡素化しているため，ループ途中の分岐条件によって終了するようなループ構造は，アクセラレータでは高速実行不可能と判断する．また，イタレーション間に依存関係があるループは，高速化できない．イタレーション間の依存関係がないことを保障するため，同一配列に対するロードとストアのアドレスの関係を解析し，ストアオフセットがロードオフセットの最小値よりも小さいこと（アドレスがデクリメントされる場合は大きいこと）を確認する．命令生成の対象とするループを抽出した後，3.1 節に述べる方法によりメモリアクセスパターンを解析し，メモリアクセス命令の最適化を行い，メモリアクセス回数と命令段数の削減を試みる．3.2 節に述べる１回目の命令生成を行い，成功すれば，各ユニットの制御情報に対応する命令を生成する．この情報を利用することにより，後述するループ分割（並列化）解析を行う．もしループ分割不可能であれば，レジスタを割り当てて，デバイスドライバに渡す命令列を生成する．ループ分割可能な場合は，分割および再命令生成を行い，命令列を生成する．

x=@(X++), y=@(Y++), z=@(Z++)
LD @(A+x), LD @(B+y), LD @(C+z)

(a) Filter detects: x, y, z are globally random.



(b) target LDs in one ARRAY stage 0

FU[0,4]: LD(mA[1024], A+x, -)
FU[0,3]: LD(mB[1024], B+y, -)
FU[0,2]: LD(mC[1024], C+z, -)

"-" indicates: local FIFO addr is not used.

(c) target MAPPING info of LDs

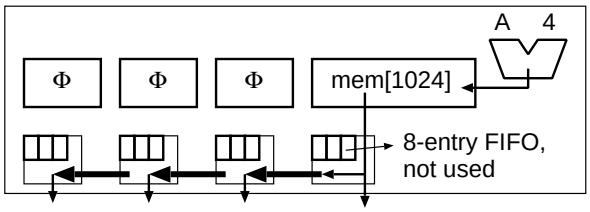
P1: Globally random

図 3 メモリアクセス・パターン（P1）.
Fig. 3 Memory access pattern 1.

LD @(A+a1), LD @(A+a2), LD @(A+a3), LD @(A+a4)

i=0, A=0	a4	a3	a2	a1
i=1, A=4		a4	a3	a2	a1
i=2, A=8			a4	a3	a2	a1

(a) Filter detects: 1. A+=4; 2. a1, a2, a3, a4 are in sequence.



(b) target LDs in one ARRAY stage 0

FU[0,4]: LD(mem[1024], A+=4, -)
FU[0,3]: LD(-, <<, -)
FU[0,2]: LD(-, <<, -)
FU[0,1]: LD(-, <<, -)

"<<" indicates the above ←

1st "-" indicates: FIFO
2nd "-" indicates local FIFO is by passed.

(c) target MAPPING info of LDs.

P2: Sequential load

図 4 メモリアクセス・パターン（P2）.
Fig. 4 Memory access pattern 2.

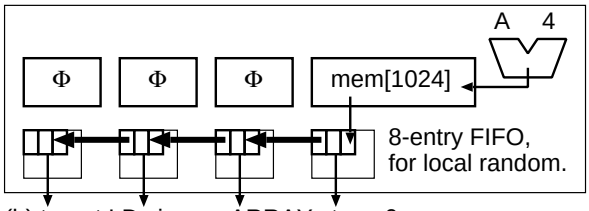
3.1 メモリアクセスパターンの解析

各行に 4 つのローカルメモリを備えるハードウェア構造を最大限利用して，アクセラレータの演算性能を最大限発揮させるために，GCC が生成するロード情報のうち，同じベースアドレスを利用するものは可能な限り同一行のユニットのロード機能として収容し，隣接ユニット間で可能な限り再利用することを目指す．この点は，LAPP が既存 VLIW 命令との互換性を維持するために複数ロード命令

LD @(A+a1), LD @(A+a2), LD @(A+a3), LD @(A+a4)

i=0, A=0	a2		a4		a3	a1	
i=1, A=4		a2		a4		a3	a1
i=2, A=8			a2		a4		a3	a1

(a) Filter detects: 1. A+=4; 2. MAX(a1..a4) - MIN(a1..a4) < 4*8



(b) target LDs in one ARRAY stage 0

FU[0,4]: LD(mem[1024], A+=4, a1)
FU[0,3]: LD(-, <<, a3)
FU[0,2]: LD(-, <<, a4)
FU[0,1]: LD(-, <<, a2)

"<<" indicates the above ←

"-" indicates FIFO

(c) target MAPPING info of LDs

P3: Globally sequential, locally random

図 5 メモリアクセス・パターン（P3）.
Fig. 5 Memory access pattern 3.

LD @(A+a1), LD @(A+a2), LD @(A+a3), LD @(A+a4)

i=0, A=0	a2		a4		a3	a1	
i=1, A=8		a2		a4		a3	a1
i=2, A=16			a2		a4		a3	a1

(a) Filter detects: 1. A+=8; 2. MAX(a1..a4) - MIN(a1..a4) < 4*8

FU[0,4]: LD(mem[2048], A+=8, a1)
FU[0,3]: LD(-, <<, a3)
FU[0,2]: LD(-, <<, a4)
FU[0,1]: LD(-, <<, a2)

"<<" indicates the above ←

"-" indicates FIFO

(b) target MAPPING info of LDs

P4: STEP +=2

図 6 メモリアクセス・パターン（P4）.
Fig. 6 Memory access pattern 4.

を同一行に対応付けることができず，行数が増大しがちであった弱点を克服する改良である．また，抽出したループのメモリアクセスパターンを分析し，イタレーション間で再利用可能なデータを利用してメモリアクセス回数を削減することも試みる．具体的には，再利用可能なローカルメモリの内容を保持したまま，命令マッピングを次行以降にシフトすることにより，データと命令の対応付けを変更し，外側ループの機能を収容する．ローカルメモリの内容を移動させずに最大限利用することで，高速化と省電力化を図ることができると考えている．すなわち，LAPP では，初

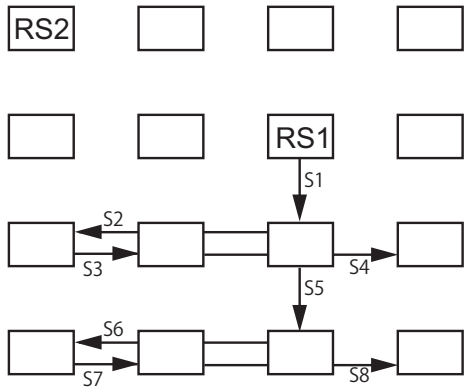


図 7　ユニットの探索.
Fig. 7　search for unit.

段においてのみ，ローカルメモリをローテーションさせて再利用を試みることが可能であった弱点を 2 次元構造全体に適用可能としたことが改良点である．

図 3 は，表 1 の P1 に対応する命令生成手順である．まず (a) は，GCC が生成する中間表現（図 2(a)）から取り出した命令レベルの情報である．これを (b) に示すようにハードウェアの構造と対応付ける．P1 の場合，各ロードはランダムオフセットであるため，各々を異なるローカルメモリに対応付けてランダムアクセスを可能とする（FIFO は使用しない）．対応付けが成功した場合，(c) に示すアクセラレータ用命令列を生成する．

図 4 は，同様に P2 に対応する命令生成手順である．全てのロードが共通のベースアドレスを使用しており，a1 から a4 までのオフセットが各々固定値である場合，ローカルメモリを 1 つだけ使用して，順次読み出しを行い，読み出しデータを各ユニットに順次転送する．最も遠いユニットでは，最も古いデータを参照することができる．この参照パターンは，FIFO を使用しても実現可能であるものの，FIFO を停止することで，低電力化を図ることができる．

図 5 は，P3 に対応する命令生成手順である．全てのロードが共通のベースアドレスを使用しつつ，a1 から a4 が固定ではなく，狭い範囲で変化する場合である．この場合も，同様にローカルメモリを 1 つだけ使用して，順次読み出しを行い，読み出しデータを放送により隣接ユニットの FIFO に格納する．各ユニットでは，FIFO に格納されている範囲のデータを読み出すことができる．なお，図 4 および図 5 において，全てのデータを 1 つのローカルメモリに収容できない場合は，複数のローカルメモリを使用して，順次，各ローカルメモリからデータを放送しつつ，FIFO に蓄積することにより，利用可能なローカルメモリの容量を増やすこともできる．

図 6 は，P4 に対応する手順である．これは，参照間隔が広いストライドアクセスに対応している．

3.2　命令マッピング

命令マッピングの際には，ハードウェアの各ユニットの使用状況を把握する必要がある．このために，メモリアクセスパターンの解析後，1 回目の命令マッピングを行って，全ユニットの使用情報を生成する．まず，命令のデスティネーションレジスタは，演算器の出力レジスタに対応付けられるので，同じソースレジスタ番号を使用する命令は，データの移動距離を最短にするために，なるべく直下のユニットに対応付ける．すなわち，デスティネーションレジスタが存在するユニットの番号に基づき，次にマップすべきユニットの位置を決定する．もし使用済みであれば，次に近い空きユニットへの写像を試みる．デスティネーションレジスタ直下のユニットが全て使用済みである場合は，さらに次の行を検査してマッピングを進める．**図 7** は，この探索の順序を s1 から s8 により示している．

FIFO を使用するロード命令については，1 つの命令をローカルメモリに付随する EAG にアドレスを供給し，メモリから読み出したデータを FIFO に伝搬するようマッピングを行う．隣接する命令については，EX1 を用いて FIFO にアドレスを供給し，FIFO から読み出しを行う．ローカルメモリからデータを読み出す命令については，まず前述の手順と同様に，マッピング可能なユニットを探索し，当該ユニットを含む行に所属するユニットの EX1 と EAG に十分な空きがあるかどうかを検査する．所要の数を収容できない場合は，引続き，次の行を検査する．マッピング可能な行に多数の空き EX1 や空き EAG が存在する場合には，他の命令と合わせて，命令マッピングを行う．

4.　ループ分割による並列化

LAPP では，初段に配置した基本プロセッサから供給されるデータを後続のアクセラレータ部分が利用し，演算結果を初段に書き戻す構成であったため，アクセラレータ部分に未使用演算器があっても，ループを分割して，スループットを 2 倍に増加させることが困難であった．本提案では，均一なリング構造とすることにより，未使用部分を最大限に利用することを狙っている．ユニットの使用状況により，マッピングした命令列を，未使用のユニットにもマッピングし，ループ分割による並列化を試みる．例えば，4 列×8 行のユニットのうち，2 行分し使っていない場合，2 行分の命令を残りの行にコピーし，演算性能を 4 倍に向上する．もちろん，列方向に空きがあれば，列方向も拡張可能である（**図 8**）．行と列の拡張を行い，命令列を生成した後に，命令マッピングを行い，最終的に，ループ分割に従って，ローカルメモリへのデータプリフェッチ情報を生成する．

5.　評価と考察

本章では，LAPP の評価に使用していた画像処理プログ

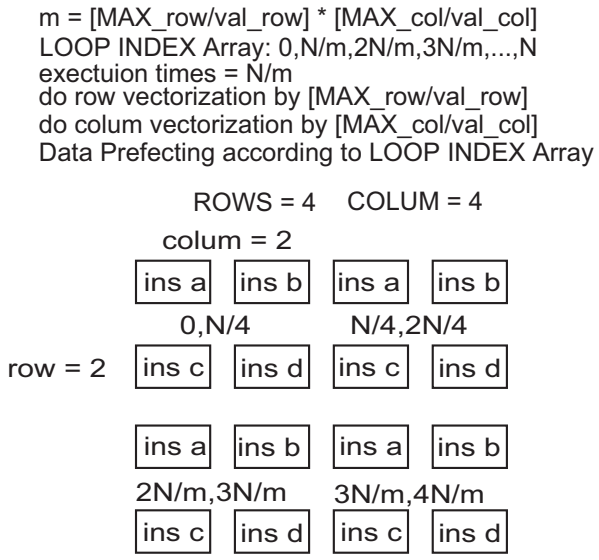


図 8　ループ分割による並列化.
Fig. 8　Parallelization by loop division.

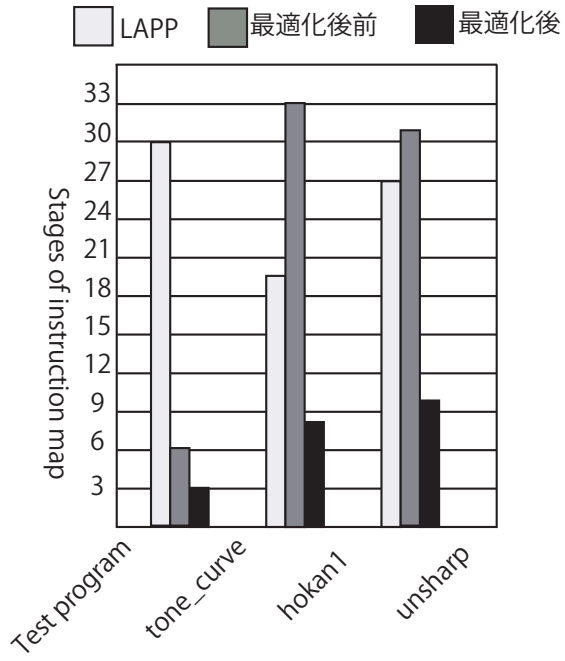


図 9　命令段数の比較.
Fig. 9　Comparison of number of rows.

ラムを用いて，提案する命令変換手法の定量的評価を行う．プログラムには，鮮鋭化（unsharp），フレーム補間の SAD 計算部分（hokan1），色調補正（tone_curve）を用いた．各プログラムについて，gcc-4.4.6 を用いてコンパイルし，生成された uncprop file の中間表現を利用して，提案手法による最適化を行った．命令マッピングに必要となる行数を比較した結果を図 **9** に示す．各行に 1 つのロードのみマッピング可能な LAPP，および，本アクセラレータ用の命令列であるものの最適化前の命令列（同様に 1 行に 1 つのロードが記述されている状態）と，提案する最適化手法を比較した．提案手法では，確実に行数を削減できていることがわかる．今後，評価対象のプログラム数を増やすことが必要であるものの，現状では，LAPP に対して平均 65%，最適化前の命令列に対して平均 60%を削減できることがわかった．さらに，全体で 3 2 行構成のアクセラレータを仮定した場合，最適化により生じた空き演算器を使用して，ループ分割を行うことができ，LAPP に比べて，2 倍から 8 倍の性能向上を期待できることがわかる．

謝辞　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），科学研究費補助金（基盤研究（A）24240005，若手研究（B）課題番号 23700060）及び JST-ASTEP（FS 課題番号 AS232Z02313A）による．

参考文献

[1] Yi Yang, Ping Xiang, Jingfei Kong and Huiyang Zhou. A GPGPU copiler for memory optimization and parallelism management. In PLDI '10 Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation, Pages 86-97, 2010
[2] 齊藤光俊, 下岡俊介, Devisetti Venkatarama Naveen, 大上俊, 吉村和浩, 姚駿, 中田尚, 中島康彦: "線形演算器アレイ型アクセラレータを備えた高電力効率プロセッサの開発", 電子情報通信学会論文誌 D, Vol.J95-D, No.9, pp.1729-1737, Sep. 2012
[3] Hao Wang, Jun Yao and Yasuhiko Nakashima. 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. In IPSJ SIG Notes 2012-ARC-199(15), 1-4, 2012-03-20.