Vol.2013-ARC-206 No.8
2013/7/31

メモリ分散型アレイアクセラレータの浮動小数点演算に関する性能考察

林 大地[1]　関 賀[1]　原 祐子[1]　姚 駿[1]　中島 康彦[1]

概要：先行研究である LAPP 上で浮動小数点演算プログラムを実行させる場合，演算器とメモリ間のトポロジに関していくつかの課題が明らかになった．本課題は主に LAPP が既存 VLIW とのバイナリ互換性を維持していることに起因している．このため，バイナリ互換性と引き替えに，浮動小数点演算プログラムを効率的に実行する新たな構成のアクセラレータを提案する．まず，局所メモリを分散配置することにより，より多くの配列を参照するアプリケーションプログラムに対応した．また，浮動小数点演算器を配置しやすい 2 段パイプライン構造とすることにより，動作周波数も維持した．さらに，横方向の配線数を削減するために構成の見直しを行った．ステンシル計算である大気シミュレータ GRAPES を用いて性能予測を行った結果，命令写像に要する段数が LAPP では 40 段であるのに対し，提案する EMAX では 15 段と半減でき，縦方向のデータ移動距離も短縮できることがわかった．

1. はじめに

近年，ステンシル計算の高速化手法に関する研究が盛んに行われている．ステンシル計算には，演算対象となるデータ量に比べて計算量が多い特徴があるため，データの再利用性を生かすアクセラレーションにより，メモリボトルネックを解消して多数の演算器を最大限活用することができる．本稿では，まず 2 章において，LAPP 上に大気シミュレータ GRAPES を実装した先行研究 [2] の概要，および，LAPP の課題について述べる．次に 3 章において，課題を解決するための具体的方策について考察し，新たな実行モデルに基づくアクセラレータ（EMAX）の実装上の工夫について述べる．最後に，4 章においてハードウェアの効率的利用の観点から評価を行う．

2. 先行研究 LAPP と大気シミュレータ GRAPES

本章では，主に画像処理の高速低電力化に特化して開発したアクセラレータ（LAPP[1]）の概要と，評価に用いる大気シミュレータ（GRAPES）の概要を述べ，LAPP に浮動小数点演算器を搭載する場合の課題についてまとめる．

2.1 GRAPES の概要

GRAPES（Global/Regional Assimilation and Prediction System）は式 1 に従ってステンシル計算を行うプログラムである。k，j，i からなる 3 重ループの最内 i ループに着目すると，各イタレーションは**図 1** に示すアクセスパターンとなる．2x2x2 キューブの頂点を除く格子点（$(i \pm 1, j \pm 1, k \pm 1)$ の計 19 点）が配列 B に，また，各格子点に乗じる係数が 4 次元配列 A に格納されている．すなわち，1 イタレーションにつき必要となるロード数は，配列 B から 19，配列 A から 18（A の中央に対応する $A_{x,i,j,k}$ は常に 1.0 であるため 1 つ省略可能）となる．37 個のロードデータに対して，乗算を 18 回，加算を 18 回適用して 1 つの結果を得る．

$$C_{i,j,k} = \sum_{\delta_1=-1,\delta_2=-1,\delta_3=-1}^{\delta_1=1,\delta_2=1,\delta_3=1} (\alpha_{x,i+\delta_1,j+\delta_2,k+\delta_3} \times B_{i+\delta_1,j+\delta_2,k+\delta_3})$$
$$\text{, where } \alpha_{x,i+\delta_1,j+\delta_2,k+\delta_3} = \begin{cases} 1, & \text{if } \delta_1 = 0, \delta_2 = 0, \delta_3 = 0 \\ A_{x,i+\delta_1,j+\delta_2,k+\delta_3}, & \text{if } \delta_1\delta_2\delta_3 = 0 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

2.2 LAPP による実行モデルと課題

図 2 に示すように，最内ループが進むにつれて i が進み，配列 B の参照範囲は i 方向に 1 要素ずれていく．また，外側ループが進むと，参照範囲は j 方向に 1 要素ずれる．すなわち，このステンシル計算を連続して行うためには，i 方向の連続データを合計 9 列（$B_{*,j\pm1,k\pm1}$）演算器に供給できればよい．また，j ループの新たなイタレーショ

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

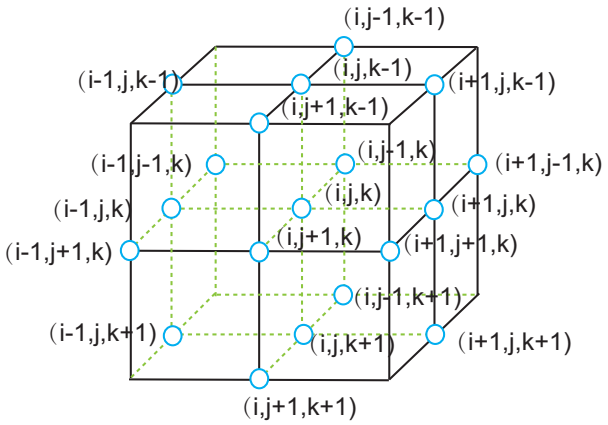


図 1　参照パターン.

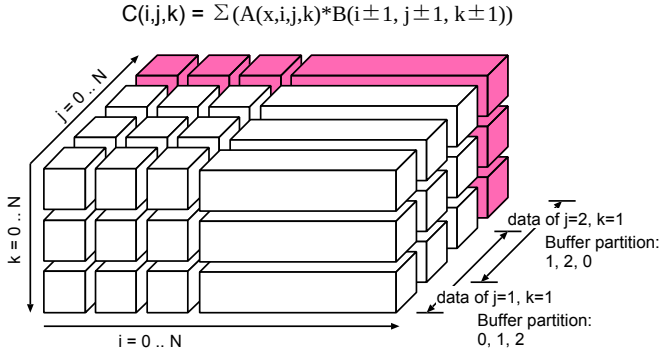


図 2　参照範囲の移動.

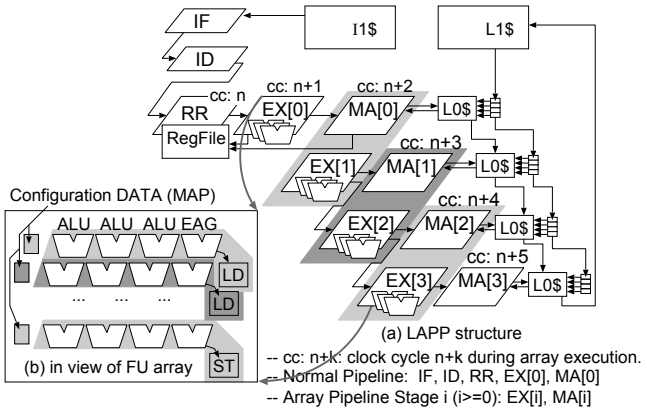


図 3　LAPP の概略構造.

ンを開始する際には，前イタレーションにて使用した連続データを合計 6 列（$B_{*,oldj,k\pm1}$ と $B_{,oldj+1,k\pm1}$）再利用でき，新たに配列 B の合計 3 列 ($B_{*,newj+1,k\pm1}$) を演算器に供給できればよい．先行研究の LAPP（Linear Pipeline Processor，図 3）を用いて性能を見積もった結果，汎用プロセッサに比べて，5.8 倍の性能向上が可能であることがわかっている [2].

さて，LAPP は，VLIW 向けの演算ユニット群を多数線形配置した構造を有する．最内ループ i を記述した VLIW 命令列を演算器に写像し，初段に配置した L1 キャッシュから狭いアドレス範囲のデータ列を後段に伝搬させながら

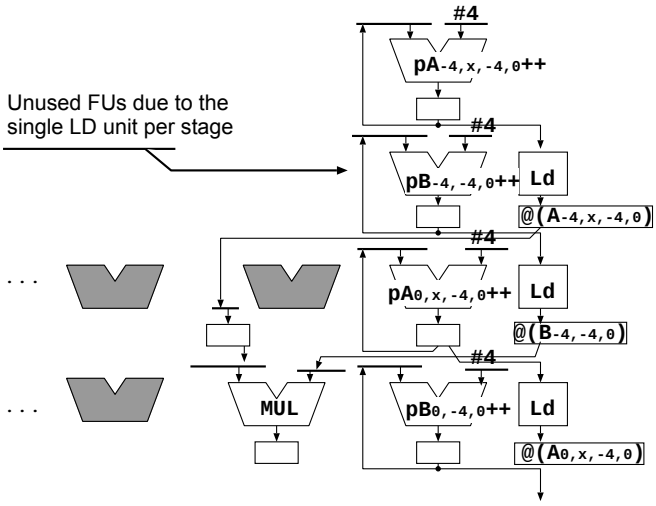


図 4　LAPP の実行モデル.

演算結果も伝搬させることにより，各イタレーションの実行結果を毎サイクル出力することができる．ただし，ハードウェア量を削減しつつ，典型的な画像処理に最適となるよう，各段に収容可能なロード命令を高々 1 としていた．GRAPES の場合，前述したように合計 37 個のロード命令の写像が必要であるため，図 4 のように縦に長い命令写像となる．2 つのロード結果に対して 1 つの乗算および 1 つの加算を行うことから，積和命令を使用した場合，1 段置きに積和命令が写像される．概ねロード命令数によって必要段数が決定され，最内ループ全体の写像に必要な段数は 40 段となる．このように，LAPP の構成では，ロード数と演算数がほぼ等しい命令列を写像する場合，演算器使用効率が極めて低くなる．さらに，浮動小数点演算器は整数演算器よりも遅延時間が大きく，LAPP の各段配置した整数・メディア演算器を単純に置き換えた場合，全体の動作周波数が半分程度にまで低下する．

以上のことから，LAPP の演算器を単純に浮動小数点演算器に置き換えただけでは，効率の良い浮動小数点演算向けのアクセラレータを構成することが難しいと言える．

2.3　GRAPES の分析

浮動小数点演算を用いるステンシル計算に適したアクセラレータの構成を探索するために，まず，GRAPES のメモリアクセスパターンの分析結果を列挙する．

(1) 4 次元配列 A からロードする 18 個の係数（$A_{x,i,j,k}$）は，最内ループの i に関して連続ではなく，x の値に依存してランダムアクセスとなる．このため，LAPP では，連続アドレスになるよう，$A_{i,x,j,k}$ に変形した後に使用している．しかし，変形後も，j ループのイターレーションにおいて，前イタレーションが使用した A を再利用できる余地がないことから，A に関するメモリボトルネックの解消は困難である．むしろ B の再利用を妨げない配置を可能とする必要がある．

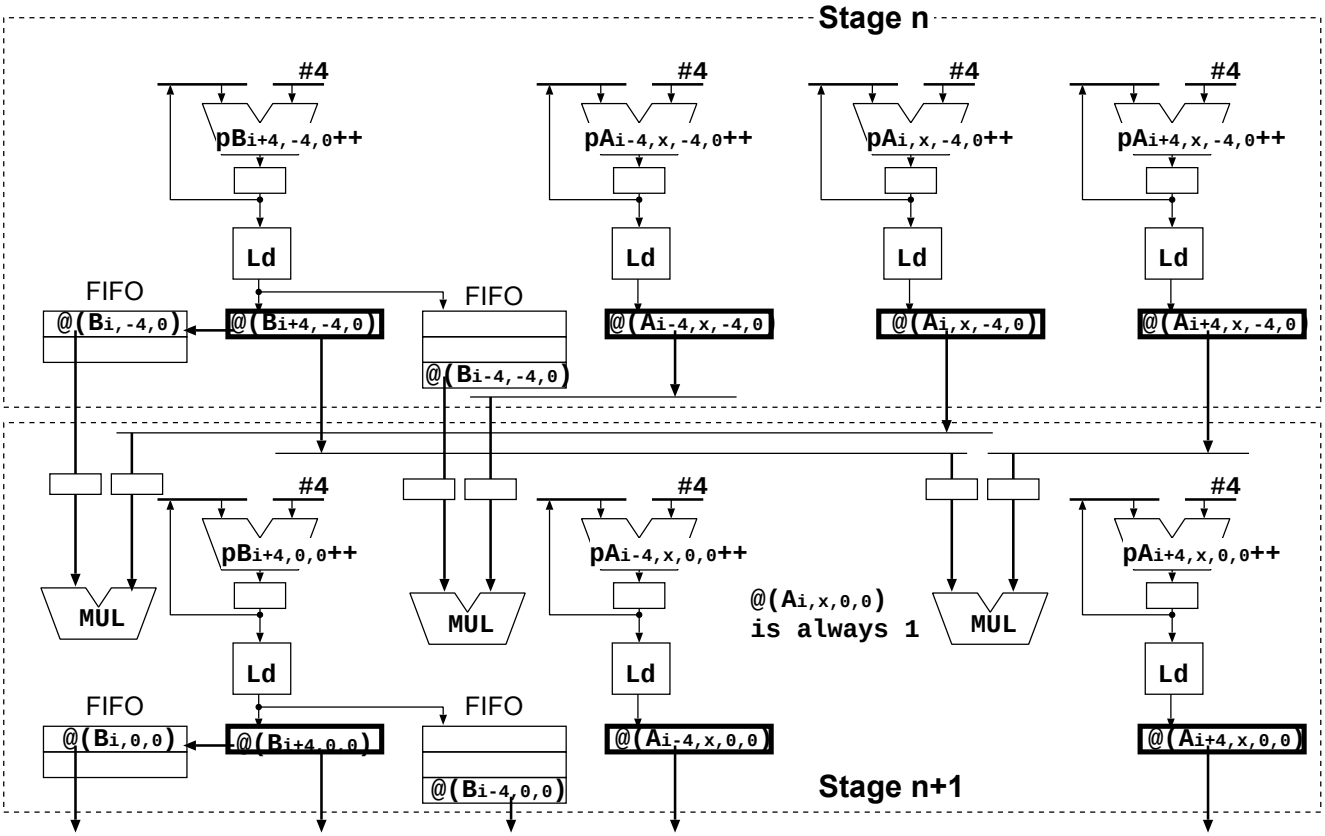


図 5　新たな実行モデル.

(2) 配列 B からロードする 19 組の $B_{i,j,k}$ は，参照順序が規則的である．前述の通り，9 組の $B_{*,j\pm1,k\pm1}$ の中からロードすることができ，次のイタレーションにおいて最大 2/3 を再利用することができる．しかし，LAPP には，9 組の配列を同時に参照しつつ，2/3 を効率良く再利用する仕組みがない．初段に集中配置したメモリから後段に供給する LAPP の基本構造には限界がある．

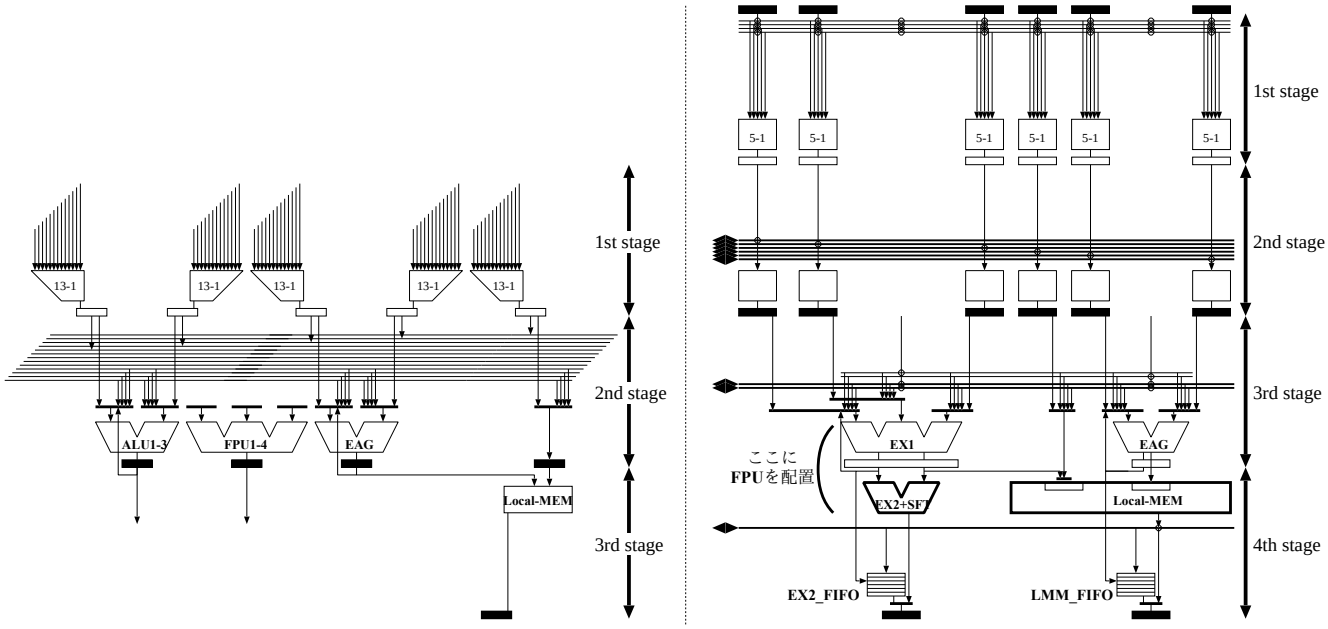
(3) 再利用が困難な A と，計画的な再利用が可能な B から 1 つずつをロードし，演算器に投入する場合，演算器の近くに 2 つのメモリを配置してデータを供給するほうが，無駄なデータの伝搬を抑制できる．また，隣接データを同時に隣接演算器に供給できる構成が望ましい．

3. メモリ分散型アクセラレータへの転換

LAPP が浮動小数点演算アクセラレータとして使い難い原因は，演算器間の接続トポロジではなく，専らメモリと演算器の接続トポロジにあると言える．LAPP のメモリ-演算器間トポロジは，既存 VLIW 命令列をそのまま利用してプログラム開発コストを低減することを目的として設計したため，このトポロジを変更することは，VLIW との互換性を放棄することを意味する．しかし，アクセラレータに関するコンパイラ技術が向上してきているため，互換性よりも潜在能力の向上を優先することとした．

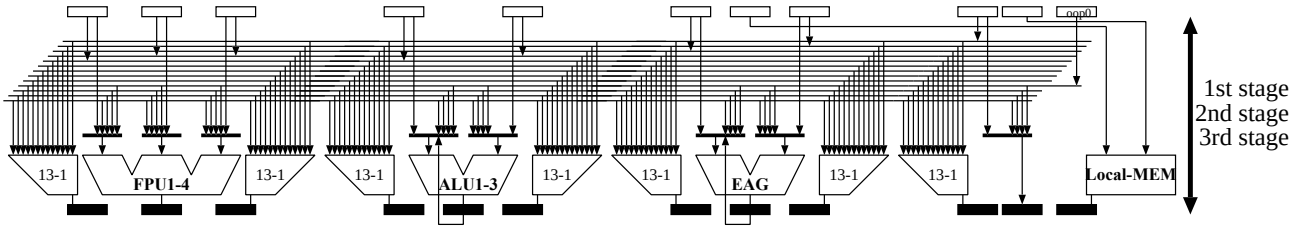
3.1 新たな実行モデルの提案

前章に述べた (1) から (3) の課題を解決する実行モデルを示す．まず，$B_{i\pm1,j-1,k}$ および $B_{i\pm1,j,k}$ と，対応する配列 A に着目する．参照するアドレスは 11 箇所あり，このうち $B_{i\pm1,j-1,k}$ の 3 箇所と $B_{i\pm1,j,k}$ の 3 箇所が各々隣接している．また，$A_{i\pm1,x,j-1,k}$ の 3 箇所と $A_{i\pm1,x,j,k}$ の 2 箇所（中央は不要）が各々隣接する．すなわち，1 つのローカルメモリに $B_{i\pm1,j-1,k}$，後段のローカルメモリに $B_{i\pm1,j,k}$ を割り当てると同時に，FIFO を経由して隣接データを水平方向に供給する仕組みが適すると考えられる．また，$B_{i\pm1,j-1,k}$ と同じ段のローカルメモリに $A_{i\pm1,x,j-1,k}$，$B_{i\pm1,j,k}$ と同じ段のローカルメモリに $A_{i\pm1,x,j,k}$ を各々割り当てることにより，演算器までの距離を最短にできる．以上のスケジューリングを適用したモデルが図 **5** である．Stage n において，4 つのローカルメモリのうち 3 つに配列 A の異なる 3 列が割り当てられており，残り 1 つに配列 B が割り当てられている．両隣の FIFO には，配列 B から読み出したデータが一時的に格納され，隣接要素を Stage n+1 に供給する．Stage n+1 では，3 つの乗算を実行すると同時に，Stage n と同様のメモリ配置により次段へデータを供給する．このように，演算器とローカルメモリを隣接させることにより，演算器の使用効率を高めることができる．

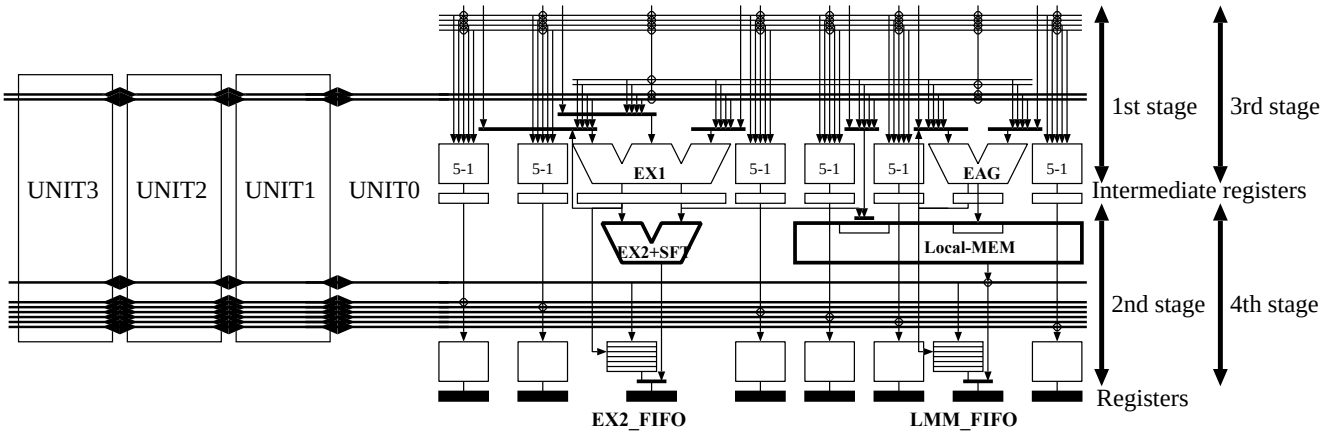


(a) LAPPのデータパス構成(folding前)　(b) EMAXのデータパス構成(folding前)

図 6　Folding 前のデータパス構成.



(a) LAPPのステージ構成(folding後)



(b) EMAXのステージ構成(folding後)

図 7　Folding 後のステージ構成.

3.2　EMAX の構成

LAPP では，ローカルメモリを初段に集中配置して狭いアドレス範囲のデータを後段に伝搬させることにより，狭い範囲であれば任意の段にロード命令を記述することができた．このため，既存 VLIW 命令列とのバイナリ互換を維持でき，命令スケジューリングの煩わしさを低減することができた．しかし，ローカルメモリの数（ポート数）を増やすためには，分散配置する必要がある．このため，バイナリ互換を犠牲にする選択をした．また，ローカルメモリのアクセス時間に比べて遅延時間の大きい浮動小数点演算器に対応するために，各ユニット内の演算器をカスケード接続に変更し，パイプライン 2 段に収容する構成とした．

ローカルメモリ数を増やすことにより，各段に必要なパイプラインレジスタ数が増加する．LAPP では，図 **6**(a)

に示すように，1st-stage において前々段の全てのパイプラインレジスタ（演算器ソースレジスタ 9 本+演算器出力レジスタ 3 本+ローカルメモリ読み出しレジスタ 1 本の合計 13 本）から 1 つを選択して演算器のソースレジスタに格納し，2nd-stage においてソースレジスタまたは前段の演算器の出力レジスタから 1 つを選択して演算を行い，3rd-stage において必要に応じて狭いアドレス範囲のデータを一次的に格納するメモリを参照する 3 段パイプラインを基本データパスとしていた．しかし，EMAX では，前々段の全てのパイプラインレジスタ（演算器ソースレジスタ 24 本+演算器出力レジスタ 4 本+ローカルメモリ読み出しレジスタ 4 本の合計 32 本）が多いため，動作周波数を下げなければ 1 ステージでの選択が困難である．このため，レジスタの選択に 2 ステージを使用する選択をした．

図 6(b) に示すように，1st-stage においてユニット内のレジスタ 8 本から 1 つを選択し，2nd-stage において隣接ユニット間のレジスタ 4 つから 1 つを選択する．3rd-stage では固定小数点算術演算または浮動小数点演算の前半（EX1），および，アドレス計算（EAG）を行い，4th-stage では固定小数点論理演算または浮動小数点演算の後半（EX2），および，ローカルメモリ参照（Local-MEM）を行う．なお，ステンシル計算特有の隣接要素参照のために，Local-MEM から横方向にバスを配置して，EX1 および EAG のアドレス情報により参照可能な FIFO（EX2_FIFO および LMM_FIFO）へロードデータを供給する構成としている．

さて，図 6 は，演算器に至る個々のデータパスをわかりやすく表現したものである．一方，**図 7** は，配線数と遅延バランスの見積もりのため，また，ハードウェア設計容易化のために，物理構造を意識して，異なる階層のデータパスを 1 つのモジュールに畳み込んだ各段の基本構成を示している．図 7(a) に示す LAPP では，互換性を維持するために，前段の全てのパイプラインレジスタから各演算器へデータを供給するために多くの配線を配置している．一方，図 7(b) に示す EMAX では，ユニット間の配線をバス構造とし，命令写像時に静的にバスを割り当てる構成に変更した．これにより，命令間に任意のデータ依存関係を許容することはできなくなったものの，静的ルーティングにより LAPP において動作していた命令列を写像できる能力を維持しつつ，配線数を削減できている．

また，LAPP では 1 ユニットを通過するのに 1 クロックを要するのに対し，EMAX では 1 ユニットが 2 ステージから構成されており，通過するのに 2 クロックを要することとなった．ただし，LAPP が有していた，各イタレーションの演算結果が毎サイクル出力される特徴は，EMAX も継承している．以上のように，EMAX は，LAPP の基本的な考え方を踏襲しつつ，バイナリ互換性と引き替えに，より少ない段数で多様なメモリ参照パターンに対応する構成とした．また，図 7 の配線数からわかるように，EMAX の構成は，LAPP において問題となっていた配線数の削減にも貢献している．Verilog により記述した EMAX を Design Compiler により合成した予備評価では，配線数を約 6 分の 1 に削減できており，同一面積の LSI に対してより多くのユニットを実装できると考えている．

@0,0	add (ri+=,4),r0		
@1,0			ld B(r0,-1280),r2
@1,1			ld A(r0+=,4),r12
@2,0		ld B(r0, 0),r5	ld B(r0, 4),r4
@2,1		ld B(r0, -4),r6	ld A(ri+=,4),r14
@2,2	fmul (r2,r12),r20		ld A(ri+=,4),r15
@2,3			ld A(ri+=,4),r16
@3,0	fmul (r4,r14),r21		ld B(r0, 1280),r8
@3,1			ld A(ri+=,4),r18
@4,0	add (ri+=,4),r0		
@4,1	fma3 (r5,r15,r20),r20		
@4,2	fma3 (r6,r16,r21),r21		
@5,0		ld B(r0,-1280),r2	ld B(r0,-1276),r1
@5,1		ld B(r0,-1284),r3	ld A(ri+=,4),r11
@5,2	fma3 (r8,r18,r20),r20		ld A(ri+=,4),r12
@5,3			ld A(ri+=,4),r13
@6,0		ld B(r0, 0),r5	ld B(r0, 4),r4
@6,1		ld B(r0, -4),r6	ld A(ri+=,4),r14
@6,2	fma3 (r1,r11,r20),r20		
@6,3	fma3 (r2,r12,r21),r21		ld A(ri+=,4),r16
@7,0		ld B(r0, 1280),r8	ld B(r0, 1284),r7
@7,1		ld B(r0, 1276),r9	ld A(ri+=,4),r17
@7,2	fma3 (r3,r13,r20),r20		ld A(ri+=,4),r18
@7,3	fma3 (r4,r14,r21),r21		ld A(ri+=,4),r19
@8,0	add (ri+=,4),r0		
@8,1	fma3 (r5,r15,r20),r20		
@8,2	fma3 (r6,r16,r21),r21		
@9,0	fma3 (r7,r17,r20),r20		ld B(r0,-1280),r2
@9,1	fma3 (r8,r18,r21),r21		ld A(ri+=,4),r12
@10,0		ld B(r0, 0),r5	ld B(r0, 4),r4
@10,1		ld B(r0, -4),r6	ld A(ri+=,4),r14
@10,2	fma3 (r9,r19,r20),r20		ld A(ri+=,4),r15
@10,3	fma3 (r2,r12,r21),r21		ld A(ri+=,4),r16
@11,0	fma3 (r4,r14,r20),r20		ld B(r0, 1280),r8
@11,1	fma3 (r5,r15,r20),r21		ld A(ri+=,4),r18
@12,0	fma3 (r6,r16,r20),r20		
@12,1	fma3 (r8,r18,r21),r21		
@13,1	fadd (r20,r21),r20		
@14,0			st r20,(ri+=,4)

図 8 EMAX による GRAPES の記述．

4. 評価と考察

これまでの議論に基づいて EMAX のクロックレベルシミュレータを開発し，ハードウェア記述と回路合成結果のフィードバックを行った [3]．実現可能性を担保した高精度シミュレータを用いて，GRAPES の命令列を EMAX 上に写像し，詳細に評価を行った．**図 8** に，EMAX 命令列を示す．第 1 列の@X.Y は，X 段 Y 列のユニットに写像する命令であることを指定している．第 2 列は EX1 および EX2 に写像する命令である．積和命令 fma3 (x,y,z),w は，3 つの入力から x*y+z を求めて w に格納する．前述のように，

表 1 GRAPES を実行できる LAPP と EMAX の比較.

	LAPP	EMAX
必要段数	40	15
搭載演算器数	160	60
演算器使用率	12.5%	33%
動作周波数比	1	2

同一段のローカルメモリから FIFO を経由してロードする場合には，EX1 に ld 命令を写像する．参照可能なアドレス範囲は FIFO の容量により決定される．第 3 列は EAG および Local-MEM に写像する命令であり，Local-MEM に対するランダムアクセスが可能である．

図 5 に示した実行モデルのうち，Stage n が@5.*に，Stage n+1 が@6.*に各々対応しており，@5.*に配置した 6 個の ld 命令の実行結果が，@6.* に配置した 2 個の fma3 命令に供給されている．LAPP では 7 段が必要であったのに対して，EMAX では 2 段に収容できている．また，j ループの新たなイタレーションを開始する際には，命令写像位置を 1 つ下にずらせることにより，@6.*において利用したデータを@5.*の命令が再利用できる．GRAPES 全体としては，LAPP が 40 段必要としたのに対し，EMAX では 15 段に収容することができた．すなわち，前述したように，横方向の配線数を削減できるだけでなく，縦方向のデータの移動距離も削減できている．

改めて，前述した（1）から（3）の課題を確認すると，以下のように解決できたことがわかる．

(1) 4 次元配列 A からロードする 18 個の係数（$A_{x,i,j,k}$）を配列 B の再利用を妨げることなく収容できている（ld A）．

(2) 命令写像をずらせることにより，配列 B を最大限再利用することが可能となっている（ld B）．

(3) 再利用が困難な A と，計画的な再利用が可能な B から 1 つずつをロードし，基本的に次段の演算器に投入できている．無駄なデータの伝搬を抑制できている．隣接データも FIFO を経由して隣接演算器に供給できている．

次に，演算器とローカルメモリの使用率を比較する．GRAPES では，37 個の ld 命令，1 個の st 命令，17 個の積和演算，2 個の乗算，および，1 個の加算を使用する．LAPP の各段は，4 つの MEDIA 演算器と 1 つの EAG を備えている．GRAPES の写像に必要な 40 段構成の場合，MEDIA 演算器（FLOAT 演算器に入れ換えることを想定）が 160，EAG が 40 と大規模なものになり，使用率は，MEDIA 演算器が 20/160，EAG が 38/40 である．一方，4 つのユニットから構成される EMAX の各段は，4 つの浮動小数点演算器と 4 つの EAG を備えている．GRAPES の写像に必要な 15 段構成の場合，FLOAT 演算器が 60，EAG が 60 である．使用率は，FLOAT 演算器が 20/60，EAG が 38/60 であり，回路規模が大きい FLOAT 演算器の使用率が 12.5%から 33%に向上した（**表 1**）．

5. おわりに

本稿では，先行研究である LAPP の基本的思想を踏襲しつつ，バイナリ互換性と引き替えに，浮動小数点演算プログラムを効率的に実行する新たな構成のアクセラレータ EMAX を提案した．局所メモリを分散配置して FIFO と組み合わせることにより，ステンシル計算への対応能力を強化し，LAPP よりも少ないハードウェアを効率的に利用できることを確認した．大気シミュレータ GRAPES を用いて性能予測を行った結果，FLOAT 演算器の使用率を 12.5%から 33%に向上できることがわかった．今後は，EMAX の詳細設計を進め，ゲート数，配線数および，消費電力について LAPP と定量的に比較する予定である．

謝辞 本研究の一部は，科学研究費補助金（基盤研究（A）24240005，若手研究（B）23700060）および JST-ASTEP（FS 課題番号 AS242Z02732H）による．また，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] 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

[2] Wei Wang, Jun Yao, Youhui Zhang, Wei Xue, Yasuhiko Nakashima, and Weimin Zheng: ”HW/SW Approaches to Accelerate GRAPES in an FU Array”, IEEE Symposium on Low-Power and High-Speed Chips 2013, Apr. (2013)

[3] 関賀，姚駿，中島康彦: ”リング接続を利用しデータ移動を最小限にするアクセラレータの提案”, 研究報告システム LSI 設計技術（SLDM）SIG Technical Reports, 2013-SLDM-159, Vol.17, pp.1-6, Jan. (2013)